1

ARMY RESEARCH INSTITUTE

Research Product 88–36

MCS 2 Database Embedded Training (ET): Procedural Findings for Command and Control Systems

November 1988

Manned Systems Group
Systems Research Laboratory

U.S. Army Research Institute for the Behavioral and Social Sciences

U.S. ARMY RESEARCH INSTITUTE FOR THE BEHAVIORAL AND SOCIAL SCIENCES

A Field Operating Agency Under the Jurisdiction of the Deputy Chief of Staff for Personnel

EDGAR M. JOHNSON
Technical Director

JON W. BLADES
COL, IN
Commanding

Technical review by

Randall M. Chambers
Dorothy L. Finley
Marshall A. Narva

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS —
2a. SECURITY CLASSIFICATION AUTHORITY —	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE —	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) —	5. MONITORING ORGANIZATION REPORT NUMBER(S) ARI Research Product 88-36
6a. NAME OF PERFORMING ORGANIZATION Applied Science Associates, Inc. 6b. OFFICE SYMBOL (If applicable) —	7a. NAME OF MONITORING ORGANIZATION Army Research Institute (PERI-SM)
6c. ADDRESS (City, State, and ZIP Code) P.O. Box 1072 Butler, PA 16003	7b. ADDRESS (City, State, and ZIP Code) 5001 Eisenhower Avenue Alexandria, VA 22333-5600
8a. NAME OF FUNDING/SPONSORING ORGANIZATION U.S. Army Research Institute for the Behavioral and Social Sciences 8b. OFFICE SYMBOL (If applicable) PERI-ZA	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER OPM-85-19 W.O. 219-026, 219-045, 319-031, 419-004
8c. ADDRESS (City, State, and ZIP Code) 5001 Eisenhower Avenue Alexandria, VA 22333-5600	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 62717 · PROJECT NO. A790 · TASK NO. 142 · WORK UNIT ACCESSION NO. 1520

11. TITLE (Include Security Classification)
MCS 2 Database Embedded Training (ET): Procedural Findings for Command and Control Systems

12. PERSONAL AUTHOR(S) Ditzian, Jan L. (Applied Science Associates, Inc.), and Witus, Gary (Vector Research, Inc.)

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 84/11 TO 87/12	1988, November	96

16. SUPPLEMENTARY NOTATION
The Contracting Officer's Representative is Dorothy Finley.

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Embedded training, ~~(ET)~~ Training system development, Maneuver Control System 2 (MCS 2), Computer-based training, (CBT) Training requirements, (5/0)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Embedded Training (ET), as an integral part of weapon system design, offers what appears to be a unique and cost-effective training capability. ET was developed for the MCS 2 system to help validate a set of embedded training development procedures being undertaken in a concurrent program, "Systems Design Concepts to Support Embedded Training (ET)." MCS 2 is a battlefield management system under development by the Army Development and Employment Agency (ADEA). This effort resulted in the generation of computer-aided instruction (CAI) courseware, but did not succeed in implementing interactive ET, in which the trainee uses the operating software under controlled conditions. The causative factors for this are documented as lessons to be integrated into the concurrent program.

Keywords;

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Dorothy L. Finley	22b. TELEPHONE (Include Area Code) (404) 791-4472 22c. OFFICE SYMBOL

DD Form 1473, JUN 86 — *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Research Product 88-36

MCS 2 Database Embedded Training (ET): Procedural Findings for Command and Control Systems

Jan L. Ditzian
Applied Science Associates, Inc.

Gary Witus
Vector Research, Inc.

Manned Systems Group
John F. Hayes, Chief

Systems Research Laboratory
Robin L. Keesee, Director

U.S. Army Research Institute for the Behavioral and Social Sciences
5001 Eisenhower Avenue, Alexandria, Virginia 22333-5600

Office, Deputy Chief of Staff for Personnel
Department of the Army

November 1988

Army Project Number
2Q162717A790

Human Performance
Effectiveness
and Simulation

FOREWORD

This report describes embedded training developed for the Maneuver Control System 2 (MCS 2), a command and control system that gives commanders information about the status of their units and enemy operations. The training focuses on database operations at the upper echelon level of the system, which deals with the organization of data inputs from lower echelons and the presentation of the results to commanders. The MCS 2 database training system is presented. The system supports six lessons that instruct soldiers in the operation of the MCS 2 database. The training is computer-based training (CBT) that uses computer-aided instruction (CAI) as the only mode of presentation. In addition, the MCS 2 database authoring system is presented, which provides a means to create and execute CAI courseware on any computer running under the UNIX operating system. The authoring system, which consists of an interactive, menu-driven courseware authoring program, an authoring language, and utility software, provides a simple language and structure for nonprogrammers to use to generate and edit courseware. Appendixes are included that present the authoring system source code, instructions for creating new training and modifying the training package, and templates of authoring code implementation for selected screen presentations.

[illegible]

EDGAR M. JOHNSON
Technical Director

MCS 2 DATABASE EMBEDDED TRAINING (ET): PROCEDURAL FINDINGS FOR COMMAND AND CONTROL SYSTEMS

CONTENTS

MCS 2 DATABASE EMBEDDED TRAINING (ET): PROCEDURAL FINDINGS FOR COMMAND AND CONTROL SYSTEMS

INTRODUCTION

Background

MCS 2

Maneuver Control System 2 (MCS2) is a project of the Army Development and Employment Agency (ADEA). MCS 2 is a computer-based Command and Control tool that improves decision making by tactical unit commanders under battle conditions. MCS 2 consists of a network of computers organized into nodes via local area networks (LANs), and netted together via packet switches. In the future, digital radios will be added to free the system from wire constraints. The operating software is the Distributed Command and Control System (DCCS), running under the UNIX operating system. DCCS integrates and provides the soldier-computer interface for a number of programs designed for specific purposes, such as: word processing, spreadsheet, database, electronic mail, videographics, and other functions.

MCS 2 is comprised of two component levels, an upper echelon and a lower echelon system. At the present time, the upper echelon system uses Hewlett-Packard (HP) computers[1], coupled with graphics presentation and recording hardware and printers. The lower echelon system is comprised of GRiD computers. The lower echelon system is designed to allow lower echelon units to input their data to MCS 2. The upper echelon system organizes these data inputs and presents the results to commanders.

MCS 2 gives commanders information about the status of their units and enemy operations. These data are presented in a format that is intended to make the information easy to use for battlefield decisions. The database function is the primary tool for this purpose. The other tools mentioned above are included to clarify presentations, improve battlefield communications, and assist in other aspects of tactical decision making.

Embedded Training for MCS 2 Database Operators

ADEA requested that the Army Research Institute for the Behavioral and Social Sciences (ARI) help in implementing embedded training (ET)

[1]Certain brands of computers and software are named in this report in the interest of completeness of description of system components. This does not imply endorsement by the Army or the Army Research Institute.

in the upper echelon subsystem MCS 2 software and hardware system. ARI has an ongoing program to define ET development and design guidelines to be applied to new Army systems and selected MCS 2 as one of the exemplar systems in which to test and verify the operation of ET development procedures.

Purpose

Project

The project was intended to accomplish several goals:

Goal 1. Develop an authoring system for MCS 2 to present training on the computers used for MCS 2.

Goal 2. Develop training for MCS 2 operators to enable training at field sites for untrained operators (acquisition training) and for trained operators who have not used the system for a long while (sustainment training).

Goal 3. Ascertain what lessons the Army should derive from this program with respect to future applications of ET.

Report

This report serves two purposes. One, it discusses the overall project, its course, and its products. Two, it serves as a guideline for ADEA to use for operating and modifying the products.

Scope

At the outset of this project, a meeting of ADEA, ARI, and the project contractors established that the focus of training for this project would be database operations in MCS 2. The primary recipient of the training is to be the upper echelon MCS 2 operator.

Initially, the target hardware and software for this demonstration was to be a Wicat computer system, using Ann Arbor data terminals, running under the UNIX System V operating system. During the project this target was changed to the HP hardware noted above, running the HP version of UNIX System V. The DCCS underwent growth and changes during the program; the project responded to these changes as far as possible.

The training was to include both computer-aided instruction (CAI) and interactive ET. CAI is the presentation of information by the computer, accompanied by elicitation of trainee responses, feedback, suitable branching through the courseware, and recording of student performance. Int^eractive ET means that the trainee exercises the actual software to receive a realistic training experience.

Organization

In the remaind of this paper:

The section entitled, MCS 2 DATABASE TRAINING, describes the embedded training that was created. It starts with an in-depth discussion of the hardware and software of MCS 2, and then discusses features and content of the training, and presents a description of the authoring system.

The section entitled, MCS 2 DATABASE AUTHORING SYSTEM, presents findings and recommendations for future ET efforts in MCS 2, and for general efforts in ET on an Army-wide basis.

Appendixes are included that present the authoring system source code, instructions for creating new training and modifying the training package, and templates of authored training code that can be used to simplify the job of training creation.

Schedule

As noted earlier, a number of changes occurred in the MCS 2 program while this training development program was under way. Figure 1 shows the initial schedule established in April 1986.

These steps were followed until September 1986, when the first coordination between MCS 2 and the authoring system developers was to take place. The planned delivery of a new version of MCS 2 software to ADEA at Fort Lewis was to be complete in November 1986. This did not occur on schedule, and the expectation of delay began to manifest itself in September. This incident in the MCS 2 program has a strong affect on the training development program as well. It interfered with the training development team's access to the new software, to the MCS 2 hardware, and to the MCS 2 software developers who were fully occupied in getting MCS 2 working as quickly as possible. Accordingly, the training development program was halted until assurance could be given that access to information, software, and hardware would be available.

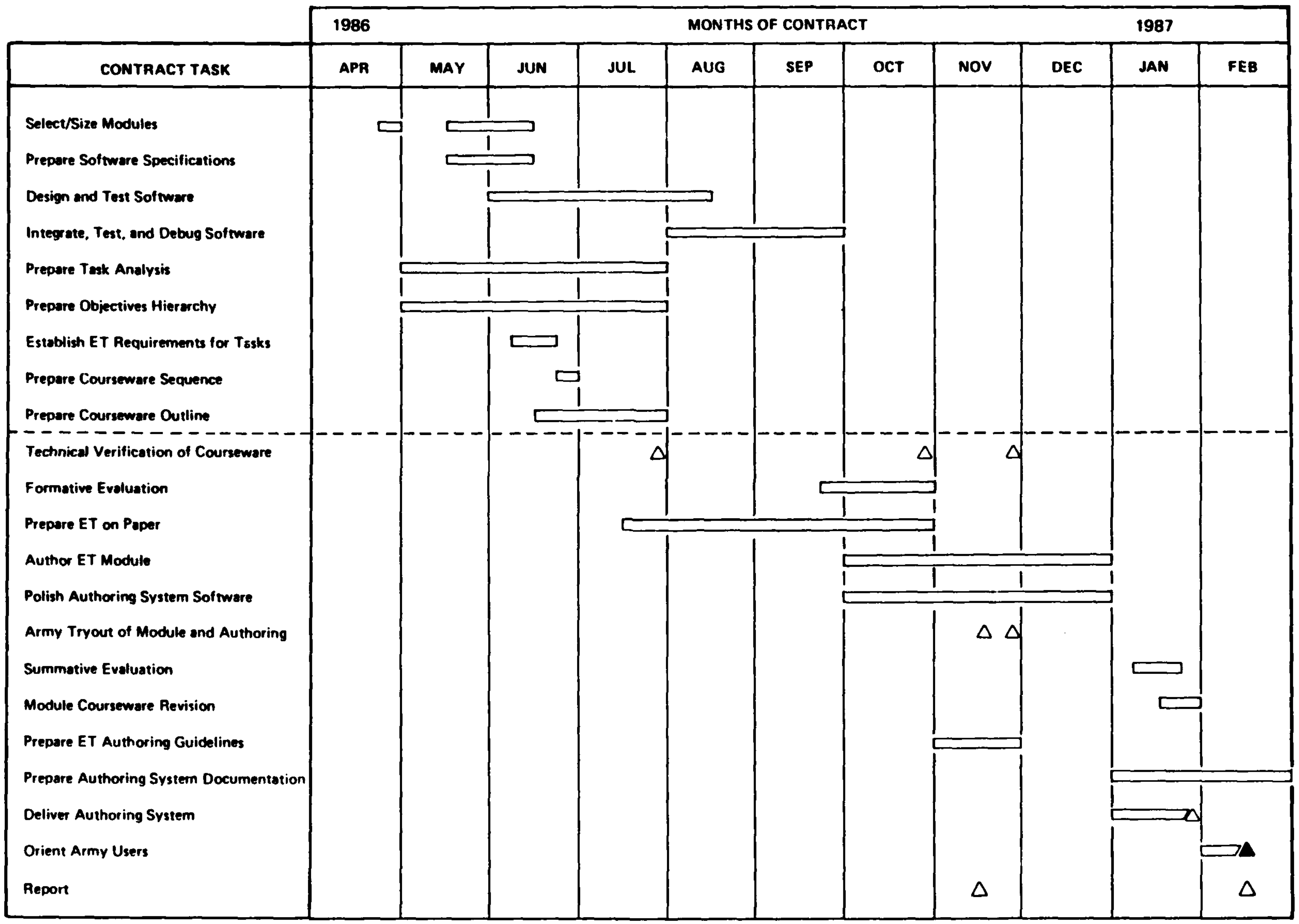

Figure 1. Initial ET contractors' internal schedule.

The program was restarted in March 1987. The change altered those schedule items below the dotted line, and the resulting schedule for these items is shown in Figure 2. The courseware was installed and demonstrated in August 1987. Comments and improvements were offered by the Army. These were incorporated into the final delivery in September 1987.

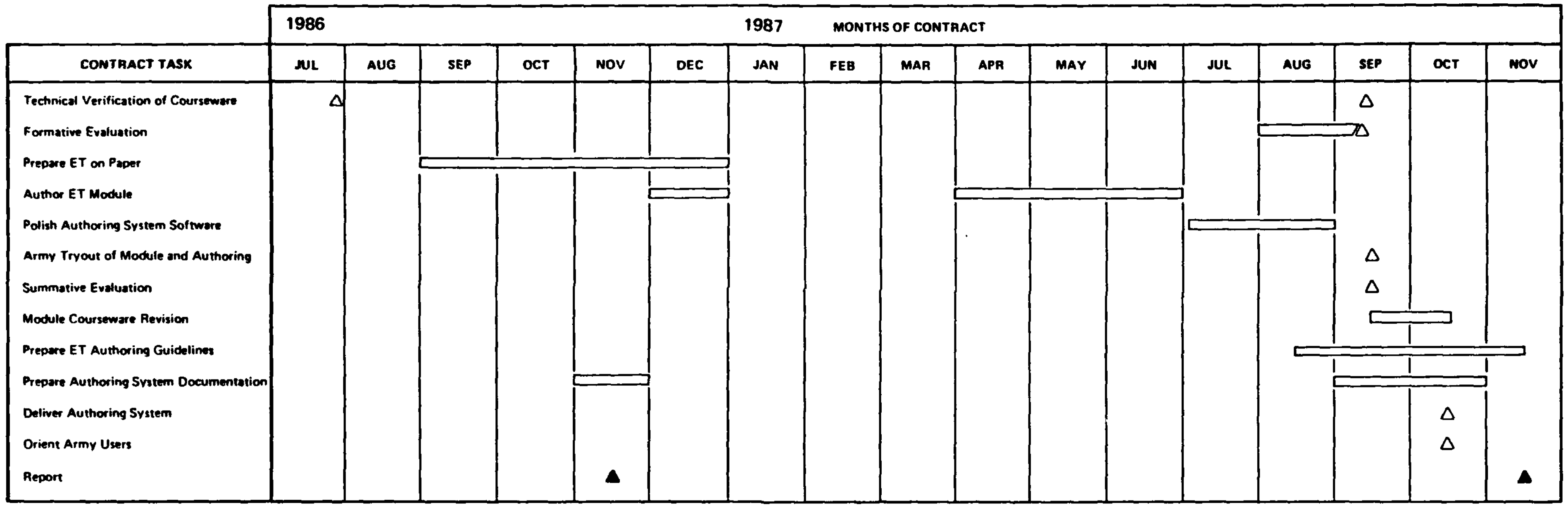

Figure 2. Resulting ET contractors' internal schedule (lower half).

MCS 2 DATABASE TRAINING

The training system that resulted from this effort is computer-based training (CBT) using CAI as its only mode of training presentation. CAI implies presentation of training material from a database of training text. CAI may or may not include features such as branching, analysis of student understanding of the material, storage of results, feedback of various sorts, and so forth. This is to be contrasted with interactive ET, which involves the use of actual operational software during the training presentation. In this case, the trainee would use the operational software under control of a main CBT program.

First, the hardware and software background for the system is presented. In the next two sections the capabilities and features of the training course are presented and the specific content of the MCS 2 database training course are discussed.

Content of MCS 2 Database Training Course

The database training course consists of six lessons that instruct a soldier in the operation of the MCS 2 database. The lessons start with an introduction to the operation of the training package itself, so there is no need for extensive briefing of the trainee prior to beginning the CBT. As long as the trainee can sign on to the training package, or is signed on by someone else, the selection of lessons and progress through the course is entirely menu driven. All interaction consists of the selection of a specified key press, initiated at the cursor at the bottom of the screen.

The first lesson is a short tutorial on how to use the training package. The trainee is given practice on interrupting training to enable him to move within the lesson or to leave the lesson.

The second lesson presents concepts to acquaint the trainee with the use of the MCS 2 terminal and the selection of choices from MCS 2 menus. A brief section on the use of the printer is included here. This latter section gives the user enough understanding of the printer to use it during training, as well as during actual database operations. The student is given an opportunity to use the printer to print out information from the training package.

The third lesson presents the overall concept of MCS 2; why it exists and its operational mission in the tactical environment. It teaches the trainee how computer nodes interact to make up the MCS 2 network, and how communications among nodes allow the relational database to do its work. The trainee also learns the relationship between upper and lower echelon nodes, and the important concept of proponency for data input. This lesson briefly presents the alternative methods for retrieving information, including Situation Reports (SITREPs) and the User Definable Reporting System (UDRS). When a user has completed this lesson he should have an appreciation of when to select a SITREP or UDRS report, and when to utilize the database query function to directly answer questions.

The fourth lesson acquaints the trainee with how data are stored and retrieved by the MCS 2 database system. This information is important because a full understanding of the underlying concepts will allow the user to develop or learn more sophisticated techniques of interaction with the system as time goes on.

The fifth lesson teaches how to retrieve specific data from the database. It guides the trainee through the use of the MCS 2 menus, and teaches how to fill in database fields, and how to initiate simple, multiple, and complex database searches.

The sixth lesson teaches how to alter database content and structure. It teaches how to change a record, and how to add or delete one. The former activity is mostly used at upper echelon nodes to put in unit commanders' assessments of their unit capabilities. Record additions and deletions are not frequent activities at upper echelon nodes, except shortly after the creation of a database for administrative purposes, such as correcting errors in the initial database structure.

When a trainee has completed the training he should be able to select the datatase function, query it for information after being given specific direction about what he is looking for, and input changes to specified fields of specified records.

MCS 2 Hardware and Software Background

The authoring language and training package prepared for upper echelon MCS 2 database training is designed to operate under the UNIX System V operating system. At the time of installation the hardware consisted of HP 310 and 320 computers, equipped with 22 and 40 mbyte hard disks, respectively. Each computer has a HP Thinkjet printer connected to it.

The training software was originally developed on, and should work with, the Wicat hardware and the associated UNIX operating system that made up the earlier version of MCS 2. Some of the training material content is not appropriate for the Wicat system, but this is restricted to the interactions with the printer and to the location of a few keys on the terminal keyboard. Since the authoring language was designed to be highly transportable, it should work with most hardware running UNIX.

Training Capabilities and Features

The training capabilities and features incorporated into the training delivered to ADEA are presented in this section. These features were devised in response to specific needs established during the development of the Database Training Course and in discussions with ADEA.

Trainee Input

The CBT created in this effort partly compensates for the lack of interactive ET by simulating some of the interactions that the trainee would have with the operational MCS 2 software. This interaction is limited by constraints of software portability and straightforwardness. For example, the trainee can press the alphanumeric keys when there is a menu from which to make a selection, but the trainee cannot press the line delete or f2 keys, since the codes for these keys are unique to the terminals used.

There are a number of presentation and response restrictions that occurred because of the inability to generate interactive ET, forcing reliance on CAI. The CAI was built entirely within the UNIX operating system language. While the software that was created and modified for MCS 2 was able to generate highlighting with reverse video, allowed full screen editing with restricted input fields, and responded to all the keys on the terminals, these effects cannot easily be achieved with the UNIX language alone. The custom software used special techniques to accomplish these ends, and these features could not be incorporated into our CAI approach. The reasons for the inability to achieve interactive ET will be discussed later.

To the extent possible, actual MCS screens are presented on the display. Some of these are modified slightly to allow room to present instructions or instructional material along with the MCS 2 screen display. It is not possible to simulate highlighting features such as reverse video or blink. Furthermore, the trainee cannot perform full screen entry or editing of data. That is, all student entries occur at the bottom of the screen. The trainee cannot fill in fields in the middle of the screen. However, where this is relevant to the procedure being taught, the trainee is taken to this point with menu selections, as he would see in MCS 2, and the entry is then simulated by the CAI. The trainee then resumes menu selection to continue the procedure.

Control Over Training Direction and Progress

The trainee can select among the six lessons at sign-on and after completing or exiting a lesson. In this way, a trainee could take Lesson 1, which teaches how to use the training, followed by Lessons 2, 3, and 4, which teach MCS 2 concepts, or the trainee could follow Lesson 1 with Lessons 5 and 6, which cover operational procedures. This allows the trainee to spend a short amount of time becoming acquainted with the aspects of MCS 2 that are most relevant to his particular responsibilities and training needs (e.g., staff officer vs. operator responsibilities, acquisition vs. refresher training).

The trainee can interrupt a lesson at most frames, except for those involving simulations of the MCS 2 software. At an interruption, the trainee is given the choice of returning to the lesson where he left it, returning to the beginning of the current lesson segment, jumping forward or back to a new lesson segment, printing the last displayed screen on the printer, or leaving the lesson.

When the trainee signs on again the lesson begins at the segment from which he left the last time.

In addition to the above control, at every instructional frame the trainee may elect to go back a frame. This process may be continued, frame by frame, but stops at quizzes and simulated interactions. In this way, the trainee may go back to check recent material, or may leave for a short time and go back a few frames for reorientation.

Questions and Feedback

Periodically the trainee is asked questions about the preceding material. Quizzes are short, from one to five questions. If the answer is correct, the program progresses to the next question or proceeds with the lesson. If the answer is not correct, the program gives feedback relevant to the particular wrong answer choice. It then branches back to the question. At this time, the question is scored as wrong, for the trainee record. The trainee must eventually make the correct answer choice to progress further through the lesson. He is returned to the question after each incorrect answer choice. The correct answer choice usually contains brief feedback that summarizes why it is the correct choice and how it differs from the incorrect alternatives.

At the end of each quiz, the trainee is told his performance score (number of questions answered and number correct) up to this time in the lesson. At the end of the lesson or if he interrupts and signs off, he is told his cumulative performance for this session in the lesson.

Trainee Records

A record of trainee performance is saved, under the name entered by the trainee at the beginning of the lesson [studentname]. If a trainee signs on a second time, the first set of records for the lesson is saved under a default file with the name [studentname].old. The data saved are: number of questions attempted in this session, number of questions answered correctly the first time they were encountered. Trainee records are accessible from the first menu by means of special commands that are not documented on this menu. The records may be printed out or deleted by means of these commands. The records are intended for training managers.

Portability

As noted earlier, and as discussed in the section on the software, portability among UNIX computer systems was an important consideration. Accordingly, the training generated herein is also as portable as possible. The file structure and files can be transferred to other UNIX systems, and can be located wherever desired at any location. What is required is that the trainee be able to get to the proper place in the file structure. The configuration manager for the system should determine this place and implement any software changes. The software coding required is quite simple for persons familiar with UNIX. Depending on the ultimate decision, two alternative implementations are possible:

1. The trainee goes to the subdirectory containing the file "oot," and type "sh oot" to begin training.

2. The file "oot" is relocated where the trainee can access it with the above command, and "oot" is altered so it sends the program to the place where the training software files and training data files are located. These files must remain in the relationship to each other that they had when installed. This relationship is documented in the section entitled, MCS 2 DATABASE AUTHORING SYSTEM.

Another aspect of portability concerns the key codes. No codes are included that are unique to a certain type or brand of terminal. Therefore, the training should run on all UNIX-based systems.

In addition, the specific procedures taught will not differ greatly between the HP and Wicat versions of MCS 2. Therefore, this training should be suitable for both versions, with perhaps minor changes for the Wicat version. This means that Wicat systems may be used for training, with the assumption that transfer of training to HP systems will be high.

MCS 2 DATABASE AUTHORING SYSTEM

Overview

The authoring system provides a means to create and execute CAI courseware on any computer running under the UNIX operating system. The courseware is transportable among different computers running under the UNIX operating system. The authoring system provides a simple language and structure for nonprogrammers to use to generate and edit courseware.

The authoring system consists of an interactive, menu-driven courseware authoring program, an authoring language, and utility software. Courses can be created and edited either by using the interactive system, or by using the UNIX command language and utilities. A course consists of a set of frames and displays. Frames contain instructions written in the authoring language to conduct input and output (I/O), to perform logical and arithmetic computations, to time events, to store student records, and to control branching to subsequent frames. Displays are screen images displayed by instructions within frames.

Language Definition

The authoring language is used to write the source frames. A source frame is a sequence of instructions. The authoring procedure includes translation of the source frames into executable frames. At runtime, the translated instructions with a frame are executed in sequence (branching and looping occurs between frames). An instruction is a series of tokens and expressions separated by blank spaces and terminated with a "newline" (i.e., a carriage return keystoke). There are 17 different types of instructions. The syntax of a type of instruction specifies the tokens, types of tokens, and types of expressions that can be used at different positions in the instruction. A token is a string of characters that does not contain embedded blank spaces. There are five types of tokens: reserved words, variable names, file names, text, and numbers. An expression is a character string that contains embedded blank spaces. There are four types of expressions: text, assignment, conditional, and regular.

The next three sections describe instructions, expressions, and tokens, respectively. Along with the text are a series of tables, to be used for reference when actually programming CAI. Following this is a section called Operating Guide, which offers some practical advice on how to program new material or modify MCS 2 CAI.

Instructions

The instructions are organized into five categories: (1) input and output; (2) logic and computation; (3) timing; (4) framing; and (5) student record keeping. Each type of instruction is described separately in the following paragraphs. Tables 1 through 5 contain an explanation of each function, an example, and the syntax definition. The syntax is defined by a generic form of the instruction that has the reserved words in their proper place, and placeholders for the author's tokens and expressions. The placeholders are angle brackets containing a description of the valid type of expression or token.

Input and Output

The author can create text displays and read user input for CAI-type interactions. Table 1 contains Input and Output Instructions. Separate instructions are provided for student I/O in MCS 2 applications interactions. The DISPLAY instruction displays text and the values currently assigned to variables. The DISPLAYFILE instruction displays the contents of a prepared file. The use of display files is strongly encouraged for more rapid system response. The PRINTFILE instruction is used to print a file on the line printer. The READ instruction reads a sequence of characters terminated by a return from the keyboard. The CLEARSCREEN instruction erases the screen.

Logic and Computation

The authoring language provides the means to assign values to variables. Table 2 contains the instructions for logic and computation operations. The ASSIGN instruction performs string and arithmetic operations, then assigns the result to the specified variable. Two instructions of the "if... then... else..." form are provided. The IF instruction tests arithmetic relations. The IFMATCH instruction tests for regular expressions in strings. If the condition evaluates to be true, then the set of instructions following the word THEN are executed. If the condition evaluates to be false, then the instructions following the word ELSE are executed. The logic and assignment instructions are used to assign frame names to the $NEXTFRAME variable. The IF and IFMATCH instructions provide the means for branching between frames.

Table 1

Input and Output Instructions

CLEARSCREEN

Function:	To erase the screen.
Example:	CLEARSCREEN
Syntax:	CLEARSCREEN

DISPLAY

Function:	To print text and the contents of variables on the same line on the screen.
Example:	DISPLAY Hello, $studentname, welcome to $coursename
Syntax:	DISPLAY <text expression>

DISPLAYFILE

Function:	To display the contents of a prepared display file.
Example:	DISPLAYFILE helpmenu
Syntax:	DISPLAYFILE <filename>

READ

Function:	To input values for one or more variables at the current cursor position.
Example:	READ $firstname $lastname
Syntax:	READ <variable> ... <variable>

PRINTFILE

Function:	To print the contents of a file on the line printer.
Example:	PRINTFILE instructionmenu
Syntax:	PRINTFILE <filename>

Table 2

Logic and Computation Instructions

<u>ASSIGN</u>

Function: To assign a value to a variable.

Examples:

```
ASSIGN $timesdone + 1 TO $timesdone
ASSIGN dog_horse_rat_goat TO $animals
ASSIGN frame4 TO $NEXTFRAME
ASSIGN $NEXTFRAME TO $lastframe
```

Syntax:

```
ASSIGN <assignment expression> TO <variable>
```

<u>IF</u>

Function: To test whether or not an arithmetic expression is true and, if it is true, to execute one sequence of instructions.

Example:

```
IF $testratio > $lastscore / 5 THEN
 ASSIGN frame3 to $NEXTFRAME
 DISPLAYFILE quizintro
ELSE
 ASSIGN frame4 to $NEXTFRAME
ENDIF
```

Syntax1:

```
IF <condition expression> THEN
   <instruction>
     . . .
   <instruction>
ELSE
   <instruction>
     . . .
   <instruction>
ENDIF
```

Syntax 2:

```
IF <condition expression> THEN <instruction>
```

Notes: Any instructions except IF and IFMATCH instructions can be used in an IF instruction. When there is only one instruction in the consequence, then the entire IF instruction can be put on one line, as shown in syntax 2.

Table 2

Logic and Computation Instructions (Continued)

<u>IFMATCH</u>

Function:	To test whether or not a regular expression is contained in character string and, if it is true, to execute a sequence of instructions. Also to test if a regular expression is contained in input and output buffers.

Example:

```
IFMATCH dog IN $animals THEN
 ASSIGN frame3 to $NEXTFRAME
 DISPLAYFILE quizintro
ELSE
 ASSIGN frame4 to $NEXTFRAME
ENDIF
```

Syntax 1:

```
IFMATCH <regular expression> IN <text, variable> THEN
 <instruction>
     . . .
 <instruction>
ELSE
<instruction>
. . .
<instruction>
ENDIF
```

Syntax 2:

```
IFMATCH <regular expression> IN
 <text, variable> THEN <instruction>
```

Notes: Any instructions except IF and IFMATCH instructions can be used in an IFMATCH instruction. When there is only one instruction in the consequence, then the entire IFMATCH instruction can be put on one line, as shown in syntax 2.

Framing

The authoring system provides facilities for packaging instructions as macros, commenting courseware, terminating the course, and executing UNIX shell commands. Table 3 contains the instructions for framing operations.

Timing

The authoring system provides facilities to time students and to have timed suspensions of course execution. The READCLOCK instruction reads the current time; it is used to determine elapsed time. The PAUSE instruction is used to suspend execution for a specified number of seconds. Table 4 contains the instructions for timing operations.

Student Profile

The authoring system provides facilities to maintain student records. Table 5 contains the list of instructions for student profile operations. Separate student records are maintained for different courses. A student record can have any number of attributes, and the attributes can have any character or numeric value. Records for new students are created with the NEWSTU instruction. The instructions WRITESTU and READSTU are used to write to and read from student records. Attributes are created simply by writing to them; they do not have to be explicitly declared.

Expressions

An expression is a string of characters that may contain blank spaces and conforms to an expression syntax. There are four types of expressions: text expressions, assignment expressions, condition expressions, and regular expressions. Examples of expressions are presented in Table 6. Table 7 shows all operators used in expressions. The following discussion mentions these operators.

Text Expressions

Text expressions are used in the DISPLAY and SEND instructions. The DISPLAY instruction causes the text expression to be displayed on the screen. The contents of variables are displayed instead of the names of the variables, however.

Table 3

Framing Instructions

<u>COMMENT</u>

Function:	To designate an internal comment in the frame.
Example:	COMMENT This line is for internal course documentation.
Syntax:	COMMENT <text expression>

<u>STOPCOURSE</u>

Function:	To terminate execution of the course.
Example:	STOPCOURSE
Syntax:	STOPCOURSE

<u>MACRO</u>

Function:	To execute a frame in place without branching.
Example:	MACRO promptframe
Syntax:	MACRO <frame file name>

<u>SHELL</u>

Function:	To execute UNIX shell commands.
Example:	SHELL cat stupro/*/*
Syntax:	SHELL <UNIX command line>

Table 4

Timing Instructions

<u>READCLOCK</u>

Function:	To read the current time (in seconds since midnight) into the variable.
Example:	READCLOCK $currenttime
Syntax:	READCLOCK <variable>

<u>PAUSE</u>

Function:	To suspend execution for a specified number of seconds.
Example:	PAUSE $delaytime PAUSE 30
Syntax:	PAUSE <number or variable>

Table 5

Student Profile Instructions

NEWSTU

Function:	To create a record for a student if no record for the student currently exists; if a record already exists, no action is taken.
Example:	NEWSTU $studentid
Syntax:	NEWSTU <variable>

WRITESTU

Function:	To update the value of an attribute in the student's profile.
Example:	WRITESTU $studentname ATTRIBUTE score VALUE 95
Syntax:	WRITESTU <filename or variable> ATTRIBUTE <filename or variable> VALUE <text, number, or variable>

READSTU

Function:	To read the current value of an attribute of the student's profile.
Example:	READSTU $studentname ATTRIBUTE score VALUE $lastscore
Syntax:	READSTU <filename or variable> ATTRIBUTE <filename or variable> VALUE <variable>

Table 6

Examples of Expressions

TEXT EXPRESSIONS:

```
Hello, welcome to the database course, $studentname
Have you used the MCS2 database before?
```

ASSIGNMENT EXPRESSIONS:

```
$elapsedtime
95
surveyframe
( 95 + $age ) * 10
```

CONDITION EXPRESSIONS:

```
$score > 15 - $lastscore
$answer = red
```

REGULAR EXPRESSIONS:

```
red
[Rr]ed
1BDE
*BDE
```

Table 7

Operators in Expressions

+	assignment operator meaning addition
-	assignment operator meaning subtraction
/	assignment operator meaning integer division
*	assignment operator meaning multiplication
(	assignment operator to establish precedence
)	assignment operator to establish precedence
%	assignment operator meaning remainder
&	assignment operator meaning logical and
|	assignment operator meaning logical or
<	relational operator meaning less than
>	relational operator meaning greater than
=	relational operator meaning equal to
!=	relational operator meaning not equal to
>=	relational operator meaning greater than or equal to
<=	relational operator meaning less than or equal to
.	regular expression symbol denoting any character
	regular expression symbol denoting the beginning of a line
$	regular expression symbol denoting the end of a line
*	regular expression symbol denoting any sequence of characters
[	regular expression symbol denoting the beginning of a character class
[	regular expression symbol denoting the beginning of a negated character class
]	regular expression symbol denoting the end of a character class
<	regular expression symbol denoting the beginning of a word
>	regular expression symbol denoting the end of a word

A text expression is simply a string of characters. It can contain variable names. The variable names must be delimited by blank spaces. Any characters including multiple blank spaces can be used. Several characters must be used with caution, in particular the dollar sign and the backslash. The first character of a variable name is a dollar sign. If a dollar sign is encountered, it will normally be interpreted as the first character of a variable name. To display a literal dollar sign, the dollar sign must be preceded by a backslash. A backslash is normally considered to be an instruction to display the following character literally, i.e., not to interpret it. To display a backslash, the backslash must be preceded by a backslash.

Assignment Expression

An assignment expression is a sequence of one or more tokens (character strings, described on Page 24) that can be evaluated to yield a text, numeric, or file name token value. Assignment expressions are used in the ASSIGN and IF instructions. Assignment expressions are evaluated and either assigned to a variable or used in a comparison.

An assignment expression that consists of a single text, numeric, or file name token is evaluated to be the token itself. An assignment expression that consists of a single variable is evaluated to be the content of the variable. Multiple token assignment expressions are algebraic expressions used to perform arithmetic computations. An assignment expression may be enclosed in parentheses with the normal algebraic meaning. Multiple token assignment expressions have the general form of an assignment expression delimited by a blank space followed by an assignment operator delimited by a blank space followed by an assignment expression. The assignment operators are +, -, *, /, (,), &, | , and %.

Conditional Expressions

A conditional expression is an expression that evaluates to be true or false. It is an arithmetic comparison between two assignment expressions. Conditional expressions are used in IF instructions. A conditional expression consists of an assignment expression delimited by a blank space followed by a relational operator delimited by a blank space followed by an assignment expression. The relational operators are =, !=, >, >=, <, and <=.

Regular Expressions

Regular expressions are used in the IFMATCH instruction. A regular expression defines a pattern of characters to search for in text. Regular expressions are used throughout the UNIX system and are described fully in the UNIX documentation of "grep" (which stands for "get regular expression"). Regular expressions are delimited by single quotes. The simplest regular expression is a character string; it defines a unique pattern. Regular expressions recognize the period as a variable standing for any single character, and the asterisk as a variable standing for any string of characters. Square brackets enclosing characters will match any one of the enclosed characters. Regular expressions are a powerful language for pattern matching and are beyond the scope of this report to describe in detail. The regular expression operators are ., $, *, [,], [, <, and >.

Tokens

A token is a character string which does not contain embedded blanks. The five types of tokens (reserved words, variable names, file names, text, and numbers) are described in the following paragraphs.

Reserved Words

Reserved words are tokens with special meaning to the authoring system. There are three classes of reserved words. One class of reserved words are the keywords used to define and parse the instruction syntax. They are the words in all capital letters in the syntax definitions. A second class of reserved words are the special operators used in assignment expressions, conditional expressions, and regular expressions. These operators are summarized in Table 7. The third class of reserved words are the special variables and file names, frame 1, ESCFRAME, and $NEXTFRAME.

Variable Names

Variable names are words used as pointers to other information. Values are assigned to variables, and the variables hold the values assigned to them. They can be used in expressions. Variables can be assigned a value in one frame, then referred to in another frame. The values of variables are lost when the course is terminated (unless they were written to the student records).

Variable names always begin with a dollar sign character ($). After the dollar sign, only alphabetic characters (a - z, A - Z), numeric characters (0 - 9), and the underscore character (_) should be used. The first character after the dollar sign must be a letter. The name must be no more than 14 characters in length. Only one variable is a reserved word: $NEXTFRAME. Each frame must either assign a value to $NEXTFRAME or else execute STOPCOURSE.

File Names

File names are words that are the names of files. They begin with an alphabetic character (a - z, A - Z). File names contain alphabetic characters, numeric characters (0 - 9), and the underscore character (_). File names are no more than 14 characters in length. There are two reserved word file names: frame_1 and ESCFRAME.

The courseware author will use file names in two cases. $NEXTFRAME is always assigned a file name (the name of the file containing the next frame to execute). The instructions to read from and write to the student profile use file names.

Text and Numbers

Text tokens are strings consisting of alphanumeric characters and punctuation marks. A number is a token consisting of only numeric characters (0 - 9). Integers of any size can be represented, but decimals are not allowed.

Operating Guide

The Operating Guide describes the procedures to use the authoring system. The first section describes how to install the authoring system. The second describes how to author courses using the interactive, menu-driven interface. The third describes how to author courses using UNIX commands. The fourth describes how to run a course, and the fifth describes how to examine student records. Appendix B contains some guidance specific to MCS 2 for adding to or modifying the CAI courseware.

Installing the System

To install the system on a new host computer the system administrator must first decide where in the directory structure the courses and authoring system will reside. On the MCS 2 system, this is in "/asa/." Two subdirectories, "uset" and "cbtcbt," must then be created using the "mkdir" command. The "uset" directory contains utility routines needed to translate and run the courseware. The files in the "uset" directory must be copied over to the new system. The "cbtcbt" directory contains the interactive, menu-driven interface. It is written in the authoring language, and is run with the authoring system utilities as any other course. Subdirectories "disp," "frame," and "exef" must be created under the "cbtcbt" directory.

Authoring Using the Menu-Driven Interface

The menu-driven interface is an interactive system that allows users to create and edit courses while requiring little knowledge of the UNIX command syntax and directory organization. It also contains on-line help explanations of the authoring language and instruction syntax.

To run the system, the user must first be in the "cbtcbt" directory. Execution is begun by issuing the command "sh ../fdrive." The user will be presented with a brief overview description of the authoring system, and a menu from which he can select a language tutorial, author courseware, or exit the system.

If the user selects the language tutorial, he is presented with two frames of description of the language organization, and then a summary of the instructions. He can then select any instruction for a more detailed description or return to the main menu.

If the user elects to author a course, he is first presented with a menu from which he can elect to: (1) display a list of existing courses; (2) select a course to edit or create a new course; (3) delete a course; (4) begin to edit a course; or (5) return to the main menu. When a new course is created, the appropriate subdirectories are created automatically. When the user elects to edit a course, he is presented with a menu from which he can elect to: (1) list existing instruction frames or display screens; (2) edit a new or existing instruction frame or display screen; (3) delete a specified instruction frame or display screen; or (4) return to the previous menu. When the user edits a display screen or instruction frame, the UNIX visual editor, vi, is automatically invoked. When new frames or displays are created, they are automatically placed in the appropriate directory. After an instruction frame is created or edited, its translation into an executable shell script is automatically created and placed in the appropriate directory.

Authoring a Course Using UNIX Command

The courseware author does not have to use the menu-driven system. He may choose to use the UNIX commands and utilities directly. This section describes the basic information needed to create a course.

The author must first create a course directory parallel with the "uset" directory (i.e., under "asa/" in the MCS 2 implementation). The name of this directory is the name of the course. The author must then create the following subdirectories: "disp," "frame," "exef," and "stupro." The author must place his display screens in files in the "disp" directory and his instruction frames in the "frame" directory. The names of these files are the names by which the displays and frames are referred to in the instruction frames. These files can be created using the visual editor, vi. The subdirectories "exef" and "stupro" are left empty. When the instruction frames are translated into executable shell script, the translations are automatically placed in the "exef" subdirectory. The "stupro" directory is used by the course to hold student records. In the database course implementation for MCS 2, the "stupro" directory should contain a subdirectory, oldstudent, as described in Appendix B.

To translate the frames into executable shell scripts, the user must first be in the "frame" directory of his course. The instruction "sh ../../ftran <file name>" will translate the designated frame. The instruction "sh ../../ftran <file name 1> ... <file name n>" will translate the list of frames. The instruction "sh ../../ftran *" will translate all frames in the directory. Each frame must be translated before it can be used in a course. Changes made to a frame will be operative only after the frame has been retranslated.

An alternative authoring approach is presented in Appendix B. This approach involves using the templates to be found in Appendix C, or by copying and modifying actual MCS 2 authoring code.

Running a Course

To run a course, the student must be in the course directory. Then to begin the course, issue the instruction "sh ../uset/fdrive." In the MCS 2 database implementation there is a controlling script that is involved by entering "sh oot." If this script is in effect, then a lesson is selected by following the menu instructions.

Examining Student Profiles

One way to examine student profiles is to operate from the UNIX system. To examine student records, the user must first change to the "stupro" directory of the appropriate course. The user can list the student subdirectories by using the "ls" command. To delete a student's records, the user must first enter the subdirectory ("cd <student id>"), delete all attribute files in the directory ("rm *"), return to the "stupro" directory ("cd .."), then remove the student's subdirectory ("RM <student_id>"). The user can obtain a list of attributes by listing the files in any student's subdirectory using the "ls" command. The user can display all of the attribute values of a specific student using the command "cat /<student name>/*." The user can display the values of a specified attribute for all students using the command "cat /*/<attribute name>." The user can display the value of a specific attribute of a specific student using the command "cat /<student id>/<attribute name>."

Alternatively, student files can be accessed from the "oot" script. If the "oot" script is running, then there are commands available to print student profiles and to erase these files. These commands are not documented on the menu. They are discussed in Appendix C. Entering +1 to +6 will print the student files for each lesson. Entering -1 to -6 will erase these files.

FINDINGS AND RECOMMENDATIONS

This effort was undertaken as a validation of a general set of procedures to develop ET in Army systems. The development of these procedures is a concurrent ARI program entitled, "System Design Concepts to Support Embedded Training (ET)." One long-range goal of this effort was to document problems with implementing ET in MCS 2 and in other Army systems. The following subsections discuss problems and present recommendations for these target areas.

Limitations of ET in MCS 2

The following are physical characteristics that limit the utility and usability of ET in MCS 2. These will be dealt with individually in this section. The order of presentation reflects the relative magnitude of importance that the system developers assign to each limitation, with the first being the most important limitation.

Disk Storage. Upper echelon MCS 2 operates with HP 320 computers acting as database hosts to HP 310 computers connected to them via LAN software and hardware. The HP 320 computers have heavy demands on their disk storage units. This demand precludes the creation of a training database resident on the computer hard disk.

Security. Interactive ET would have required a duplicate database because trainees cannot be allowed to alter the actual database. This is one reason for the large disk storage requirement for interactive ET.

Response Time. There are two factors that contribute to the slow (5 seconds) response time to a trainee input. One of these is that the shell script which runs the training is not as quick a processing language as other techniques. The second is that the HP 320 computers are hosts for up to four HP 310 computers in a LAN, as well as being involved in communications with other computers via a packet switch. One alternative is to load the training package directly onto an HP 310 computer, and to use it there.

General Implications for ET in Army Systems

This section addresses the future practical development of ET in the Army as part of the training package for a system. It represents a compendium of "lessons learned" during this effort that can be generalized to other Army procurements of ET. The recommendations are stated as positively as can be, but, as with most lessons, more is learned from mistakes than from doing things right. These lessons are not to be viewed as negative statements about any individual, agency, or company. Instead, they are simply the authors' perception of how the Army ought to pursue ET in the future, based on the particular set of experiences that pertained during this effort.

Integrate ET Into Hardware and Software from Concept Development

ET development and system development should be integrated into one acquisition program. Functional requirements, hardware suite configuration, and software architecture should take into consideration the total system, including the ET component.

Integration means including considerations for ET in the design tradeoffs for the system. In this program the UNIX system was modified by the operational software developers to reduce the amount of disk space it occupied. At one point it appeared that the resulting operating system would not support the authoring and presentation language. At another point, it appeared that training would have to be loaded from floppy disks for each session. This would have created a problem in terms of the amount of time and effort it would take to begin training. Finally, the disk storage problem that is discussed several times in this report should be considered in terms of its effect on ET speed and security of the real database during training interactions.

Ensure Adequate Hardware Capacity to Accommodate ET

Most operational systems are designed only with the capacity to handle the programs and data of normal operation. It is difficult to justify the purchase of excess capacity. Yet ET implementation requires system capacity in excess of that required for regular operations. This capacity must be acquired and dedicated to ET (not reallocated to operational system needs).

As noted above in the limitations section, there was inadequate free disk capacity in the MCS 2 system to accommodate the extra database that would be required to implement full interactive ET. The extra database is required to maintain security of the actual MCS 2 database. In addition, the ET code takes up space as well. The only way to have foreseen and planned for these contingencies was to have included ET in system planning.

Develop Close Working Relationships Among Training, Software, and Hardware Developers

As a system concept matures into actual hardware and software, it undergoes many changes. The ET developer must stay abreast of all changes to evaluate their impact on both ET implementation and content of training. The only practical way to do this is to maintain close relationships among all the players on the development team.

Changes usually go through a cycle of contemplation and proposal, followed by consideration of impact, followed by implementation of some modification of the original proposal. Close coordination is needed among the ET courseware developer, the ET implementer, and the system developer. The ET developers should be apprised of all changes as they are contemplated, in order to make inputs to the tradeoff process and to make adjustments in the training product.

This lesson has implications for the ET developer as well as the prime system developers. To be able to take advantage of this informational input, the ET development program must have schedule, resource, and approach flexibility to adapt to changes in the design and schedule of the operational system.

A second part of this coordination involves the exchange of source code and software documentation for operational software between the system developers and the ET developers. Additionally, documentation, especially at an early stage of development, can rarely stand alone. The ET developers need access to the operational system software developers themselves, in order to gain a timely and complete understanding of both the content of the software and its implications for further software development.

Ensure Sufficient Development of Authoring and Presentation Tools Prior to Designing Training

In order to meet deadlines, it is tempting to begin development of course materials as soon as objectives and courseware outlines have developed. This is fine, as long as the authoring and presentation tools or language have been well developed. In the present effort, the time schedule required the simultaneous development of both authoring and training content, followed by a rapid integration process. The result is that not all features of the authoring language are exploited to the fullest extent.

The proper mode of development is to fully establish the authoring product, and to demonstrate this product with simple training content. This permits an initial evaluation of the authoring and presentation tools independent of the training content.

Ensure Sufficient Schedule Flexibility and Resources

Sufficient time and dollar resources must be available for the ET program and the contingencies of changes in system design and schedule.

Cost estimation for the design and implementation of ET for command and control systems will become easier and more accurate when more historical data are available.

Authoring System vs. Courseware

There are at least two approaches to implementing that ET which has courseware in addition to a noninteractive stimulus presentation (as is found in "raid" tapes, for example). Each presents different organizational and technical issues. This section presents a discussion, rather than a recommendation, because it is not yet well understood when which of the two approaches is the preferred approach. If, however, fielding of the training system concurrent with the end item equipment is a requirement, then the first approach must be used.

Both approaches involve using an authoring system. This is the approach taken in this program. The courseware developers are insulated from the operational system by the authoring system. This approach frees them from detailed involvement in the system development, but insulates them from an appreciation of the limitations and capabilities of the authoring system.

If the development of the authoring and operational system are performed in parallel, then coordination requirements are extensive and resource intensive.

There is a technological risk involved in development of a general purpose authoring system that can "fool" the operational software into operating as if it had received regular operator inputs. The risk is that it will be very difficult to succeed in fooling the operating software. Some of this difficulty stems from the volatility of the operating software. Only close coordination can ensure that the training system tracks the operational system. It is necessary both to "fool" the operating system, yet leave the ET program in control. With an unsophisticated authoring system this can require extensive programming effort on the part of the training courseware creators.

An alternative approach is to develop software code to implement the courseware as part of the post-deployment software support. Because this software development is specific and can be performed by personnel with the knowledge and authority to change the operational software, it is possible that the technological risks will be lessened. On the other hand, this approach engenders several consequences:

1. ET will lag initial system test and fielding by a great deal of time.

2. Hardware will be committed to ET but will lie unused during initial system development. These resources are open to "raiding" by the system developers. In addition, changes in requirements by ET cannot be reflected in the overall

operational system development cycle, because they may not be realized until the ET development is under way, beginning at post-deployment time.

3. There will be no opportunity to exchange ET requirements with system requirements in tradeoffs, because the system will be frozen before work progresses on ET. This may result in the inability to implement some courseware.

ET Interfaces to Operational MCS 2 System Software

If ET courseware is to employ operational software, then some form of interface between the ET system and the operational system is needed.

Interactive ET would have been a valuable addition to the training, if it could have been realized. The following discussion points out the benefits of interactive ET. As we have described in other places in this report, interactivւ ET means that the actual equipment and software are exercised.

There are seven principles that define excellent interactive ET. These are presented here with information to help relate them to MCS 2 database training.

1. Operational input data are generated. In this case, the operational input signals are the proper screens to be displayed to the user, along with the correct data from the training database.

2. These data operate through the operational equipment to generate normal displays. In this case, the normal displays are the screens with the data in them.

3. Data are presented to realistically depict what would occur in an operational exercise of the system in real events, including errors. This would be actual data from a realistic database, with some data to be corrected.

4. Operators and maintainers perform their normal tasks in response to simulated mission inputs. In the case of MCS 2 this means that the operators use the system with the same keystrokes as if it were actually operating.

5. Assesses and records operator performance and reacts to that performance as the real threat would, while providing realistic feedback as the actual system would. This means actual system responses to keystroke inputs.

6. Measures performance and records that measurement for individual feedback after a session and semi-permanent records for assessments over time.

7. Permits instruction on job-related tasks in addition to the strictly operational tasks. For MCS 2 this would include setup and teardown tasks.

These principles are theoretical, some of them are rather demanding of resources, and others can only be striven for, not achieved completely.

Depending on the strategy for computer-based instruction, different types of interfaces can be used. From the perspective of software engineering, there are three alternative approaches to interface the ET and operational systems:

1. Training courseware separate from the operational software simulating the user interactions of the operational software, but without use of any of the operational software. This amounts to CAI which emulates the operating software.

2. Training courseware separate from the operational software, but capable of invoking software transported from the operational system to simulate limited controlled interaction. This amounts to CAI which invokes a special copy of the operating software.

3. Training courseware fully integrated with the operational system such that the courseware can run the operational software and control interactions between the student and the operational software.

The alternatives have different costs, strengths, and limitations. They differ in terms of the capabilities they provide for practice, demonstration, and evaluation. The alternatives are listed in order of increasing cost, complexity, coordination requirements, and capability for controlled interaction. In selecting an alternative, one must balance the capabilities required to satisfy the training requirements with the cost and coordination requirements for implementing the training program. Coordination is required between the developers and documenters of the operational system on the one hand, and the training courseware and authoring system developers and implementers on the other hand. All three options require that the actual operation and user interactions are correctly and completely documented for the courseware development.

The first alternative simply requires that the I/O screen designs, protocols, controls, device parameters settings, and environmental variable settings be thoroughly and correctly documented. The burden is then put upon the courseware to simulate those portions of the operational interactions necessary for instruction, practice, and evaluation.

The second alternative requires that the software programs within the operational system be well documented relative to their role and contribution flow of user activities, that the I/O for each program be compartmentalized, that data requirements and environmental variables setting and expectations be well defined, and that data and control passed between programs be well documented. This approach enables the courseware to run the component programs of the operational system independently to simulate real interaction.

A third option to authoring systems or courseware approaches requires that training routines be provided to enable the training software to insert itself between the user and the operational software. The principles on Pages 35 and 36 state the general functional requirements for controlled interaction with the MCS 2 operational system. This statement was prepared based on how training could interact with MCS 2 database operations. The functional requirements are general in the sense that they may need to be modified to suit the particular needs of a training program and to suit the constraints of the operational hardware and software. Additional instructions may be needed for a particular application. Specifying the functional requirements from these general requirements requires the participation of the courseware developers, the operational software developers, and the authoring system developers. After the functional requirements are specified, interface software specifications and code are produced by the authoring system developers working with the developers of the operational hardware and software.

For purposes of clarity of exposition, the functional requirements are organized into 15 distinct instructions and are presented in Table 8. The packaging of the functions could be tailored to the needs of a particular application. The instructions define capabilities to invoke the operational software, to send data to it, and to receive data from it. They define capabilities to store and restore operating system environments and terminal states. They define capabilities to search and store I/O streams and to synchronize timing between the courseware and the operational software. They define capabilities to accept input streams from a terminal and present output streams on a terminal. In short, the training system would be inserted between the operating software and the user. All the users' interactions with the system could be monitored or restricted.

Table 8

Functional Requirements and Instructions

APPEND <buffer 1> <buffer 2>

This function appends the contents of the first buffer to the contents of the second buffer. The function is useful for keeping a sequential record of student inputs and system outputs for later testing.

CLEAR <buffer>

This function empties the named buffer.

INPUT <device> <mode> <buffer>

This function takes input from the specified device operating in the specified mode, and appends the input to the named buffer. The implementation of this function is highly system specific. In particular, the modes must correspond to the modes used in the operational software.

PRESENT <device> <mode> <buffer>

This function presents the content of the named buffer on the named device in the specified device mode. The implementation of this function is highly system specific. In particular, the modes must correspond to the modes used in the operational software.

RECEIVE <device> <mode> <buffer>

This function is used immediately after a SEND instruction. Output from the operational software to the specified device in the specified mode is redirected to the specified buffer.

RESTORE_DEVICE_STATE <device> <buffer>

This function restores the state of the terminal device using the contents of the named buffer. The device state must previously have been stored in the buffer. The state of a terminal includes both the data presented on the screen and, for smart terminals, the internal memory state (e.g., protected screen regions).

Table 8

Functional Requirements and Instructions (Continued)

RESTORE_ENVIRONMENT <buffer>

This function restores the operating system environmental parameter to the values previously stored in the named buffer. The RESTORE_ENVIRONMENT and STORE_ENVIRONMENT instructions are used to avoid conflicts between the setting used by the courseware and by the operational software.

SEARCH <buffer> <pattern> THEN <instructions>

This function searches the named buffer for the specified pattern and, if a match is found, executes the following instructions. The syntax for specifying the pattern must address the issues of representing nonprinting characters such as control characters and escape sequences since they will be in both input and output streams. Consequently the standard UNIX regular expression syntax is not applicable unless extended. Care must be exercised in the pattern matcher since different terminal types will generate and receive different escape sequences to perform the same functions, hence it will be necessary to reference the TERMCAP tables. Different pattern syntaxes will be needed for buffers containing character data and graphics data.

SCREEN <device> <mode> <data>

This function sends the data to the specified device in the specified mode. A syntax for the data must allow control characters and escape sequences for screen control to be sent to terminals in text mode. The syntax for the data must allow graphics data to be sent to devices in graphics mode.

SEND <device> <mode> <data>

The function of SEND is to pipe data from the course to the operational software. The device refers to the normal source of input being emulated by the course. The mode specification may or may not be relevant, depending on the device. The data refers to the information to transmit. The type of data depends on the device, e.g., a terminal will send characters but a graphics input device will send graphics control codes. The form of the data specification must support nonprinting keyboard characters (control codes and escape sequences) in a standard way, independent of the actual terminal type. The data must be interpreted appropriately through TERMCAP entries and system environmental variables before being sent over the pipe. The data should be able to be contained in a buffer, or explicitly entered with the instruction.

Table 8

Functional Requirements and Instructions (Continued)

SETUP <batch file name>

The function of SETUP is to invoke the operational software as a subordinate process with variable settings and device assignments as specified in the batch file. Different batch files would be needed to invoke the operational software with different parameter settings. This instruction would also establish the bidirectional pipe interfaces between the course and the operational software. The specifics of setting up the bidirectional pipes depend on the I/O modes of the operational software.

STORE_DEVICE_STATE <device> <buffer>

This function reads and stores the state of the terminal device in the named buffer. The specifics of this instruction depend on the hardware and the use of the hardware. For some hardware, it may not be possible to read the device state. In this case it would be necessary to record the data stream that created the device state. The device state includes both the data displayed on the screen and, for smart devices, the data in internal device memory.

STORE_ENVIRONMENT <buffer>

This function stores the current setting of the operating system environmental parameters so that they can be restored at a later time.

TERMINATE <batch file name>

This function performs a "natural" termination of the operational software running as a subordinate task. It also closes the bidirectional pipes opened by SETUP for communication.

WAITFOR <device> <mode> <pattern>

This function is used after data is sent to the operational software to pause the course while the operation software executes. The wait ends when the specified pattern appears in the output stream to the specified device in the specified mode. The specifics of this function are highly dependent on the hardware and software of the operational system.

REFERENCES

Ditzian, J. L., Vitus, G., & Rainaldi, W. Upper echelon MCS 2 database embedded training: Recommended courseware and authoring system. Draft Interim Report. Alexandria, VA: U.S. Army Research Institute for the Behavioral and Social Sciences.

APPENDIX A

AUTHORING SYSTEM SOURCE CODE

"cfrmchk" Frame in Menu-Driven Authoring System

```
COMMENT  cfrmchk frame
COMMENT  invoked by courseme

COMMENT This frame simply checks if the course name chosen
COMMENT exists and if it doesn't it returns to course menu


ASSIGN frammenu TO $NEXTFRAME

SHELL if (test ! -d ../$course) then

DISPLAY
DISPLAY  No course has been selected. You must choose a course prior
DISPLAY  to authoring any frames.
DISPLAY  Please select a course upon returning to the course level
menu.
DISPLAY
DISPLAY  Press return to continue:

READ $dum

ASSIGN   courseme TO $NEXTFRAME

ELSE

ASSIGN frammenu TO $NEXTFRAME

ENDIF
```

"chkcours" Frame in Menu-Driven Authoring System

```
COMMENT  chkcours frame
COMMENT  invoked by selcours

COMMENT This frame simply checks if the course name chosen
COMMENT exists and if it doesn't it calls crecours.


ASSIGN courseme TO $NEXTFRAME

SHELL if (test ! -d ../$course) then
ASSIGN   crecours TO $NEXTFRAME
SHELL fi
```

"courseme" Frame in Menu-Driven Authoring System

```
COMMENT  courseme frame
COMMENT  invoked by cfrmchk, crecours, frame_1, frammenu, lstcrs,
COMMENT   rmcrs, and zapcrs

CLEARSCREEN

DISPLAYFILE leadblnk

DISPLAY  This is the menu of course authoring selections:

DISPLAY
DISPLAY  1) List existing courses
DISPLAY  2) Select course to edit or create
DISPLAY  3) Author frames of selected course
DISPLAY  4) Delete a course
DISPLAY  5) Exit - return to parent menu
DISPLAY
DISPLAY Enter choice:

READ $choice
COMMENT DISPLAY choice is $choice

ASSIGN courseme TO $NEXTFRAME

IFMATCH '^1$' IN $choice THEN
    ASSIGN lstcours TO $NEXTFRAME

ELSE IFMATCH '^2$' IN $choice THEN
    ASSIGN selcours TO $NEXTFRAME

ELSE IFMATCH '^3$' IN $choice THEN
    ASSIGN cfrmchk  TO $NEXTFRAME

ELSE IFMATCH '^4$' IN $choice THEN
    ASSIGN rmcrs TO $NEXTFRAME

ELSE IFMATCH '^5$' IN $choice THEN
    ASSIGN frame_1 TO $NEXTFRAME

ELSE
    MACRO reenter
ENDIF
```

"crecours" Frame in Menu-Driven Authoring System

```
COMMENT  crecours frame
COMMENT  invoked by chkcours

CLEARSCREEN

DISPLAYFILE leadblnk

DISPLAY
DISPLAY  Creating new course -- $course
DISPLAY
DISPLAY  Note that this includes the creation of it's subordinate
DISPLAY  directories consisting of the following:
DISPLAY
DISPLAY  disp
DISPLAY  exef
DISPLAY  frame
DISPLAY  stupro
DISPLAY
DISPLAY  Please wait while the course structure is being created.

SHELL mkdir ../$course
SHELL mkdir ../$course/buf
SHELL mkdir ../$course/disp
SHELL mkdir ../$course/exef
SHELL mkdir ../$course/frame
SHELL mkdir ../$course/stupro

SHELL cp ../uset/frame_1 ../$course/frame
SHELL cp ../uset/escframe ../$course/frame

ASSIGN courseme TO $NEXTFRAME

DISPLAY
DISPLAY  Enter return to continue:

READ $dum
```

"eddisp" Frame in Menu-Driven Authoring System

```
COMMENT  eddisp frame
COMMENT  invoked by frammenu

CLEARSCREEN

DISPLAYFILE leadblnk

DISPLAY
DISPLAY  Please enter display file name to be edited:

READ $dispname

SHELL vi ../$course/disp/$dispname


DISPLAY  Please hit return to continue:

READ $dum

ASSIGN frammenu TO $NEXTFRAME


Jan>
```

"edframe" Frame in Menu-Driven Authoring System

```
COMMENT  edframe frame
COMMENT  invoked by frammenu
CLEARSCREEN
DISPLAYFILE leadblnk
DISPLAY
DISPLAY  Please enter frame name to be edited:
READ $framename

SHELL vi ../$course/frame/$framename

DISPLAY
DISPLAY  Now translating frame -- $framename
DISPLAY

SHELL cd ../$course/frame

SHELL sh ../../uset/ftran $framename

SHELL cd ../../cbtcbt

DISPLAY
DISPLAY  Please hit return to continue:
READ $dum
ASSIGN frammenu TO $NEXTFRAME
```

"exitfr" Frame in Menu-Driven Authoring System

```
COMMENT exitfr frame
COMMENT invoked by frame_1

DISPLAY
DISPLAY   The End.
DISPLAY

STOPCOURSE
```

"frame_1" Frame in Menu-Driven Authoring System

```
COMMENT the first frame must be named 'frame_1'
COMMENT invoked by langinst and courseme

CLEARSCREEN

DISPLAYFILE  introtxt

DISPLAY  Selection:

READ $key

ASSIGN frame_1 TO $NEXTFRAME

IFMATCH '[aA1]' IN $key THEN ASSIGN langtut TO $NEXTFRAME

IFMATCH '[bB2]' IN $key THEN ASSIGN courseme TO $NEXTFRAME

IFMATCH '[cC3]' IN $key THEN ASSIGN exitfr TO $NEXTFRAME

ASSIGN nocourse TO $course
```

"frammenu" Frame in Menu-Driven Authoring System

```
COMMENT  frammenu frame
COMMENT  invoked by cfrmchk, eddisp, edframe, lstdisp, lstframe,
COMMENT   rmdsp, and rdfrm

CLEARSCREEN

DISPLAYFILE leadblnk

DISPLAY  This is the menu of frame authoring selections:

DISPLAY
DISPLAY  1) List existing frames
DISPLAY  2) List existing displays
DISPLAY  3) Edit a frame file
DISPLAY  4) Edit a display file
DISPLAY  5) Delete a frame
DISPLAY  6) Delete a display file
DISPLAY  7) Exit - return to course level menu
DISPLAY
DISPLAY Enter choice:

READ $choice
COMMENT DISPLAY choice is $choice


IFMATCH '^1$' IN $choice THEN
    ASSIGN lstframe TO $NEXTFRAME

ELSE IFMATCH '^2$' IN $choice THEN
    ASSIGN lstdisp TO $NEXTFRAME

ELSE IFMATCH '^3$' IN $choice THEN
    ASSIGN edframe TO $NEXTFRAME

ELSE IFMATCH '^4$' IN $choice THEN
    ASSIGN eddisp  TO $NEXTFRAME

ELSE IFMATCH '^5$' IN $choice THEN
    ASSIGN rmfrm TO $NEXTFRAME

ELSE IFMATCH '^6$' IN $choice THEN
    ASSIGN rmdsp TO $NEXTFRAME

ELSE IFMATCH '^7$' IN $choice THEN
    ASSIGN courseme TO $NEXTFRAME

ELSE
    MACRO reenter
ENDIF
```

"langinst" Frame in Menu-Driven Authoring System

```
COMMENT  langinst frame
COMMENT  invoked by langtut

DISPLAYFILE langtxt
DISPLAY
DISPLAY  Enter selection (or 'X' to return to the main menu):
READ $k
IFMATCH '[aA]' IN $k THEN DISPLAYFILE iassign
IFMATCH '[bB]' IN $k THEN DISPLAYFILE iclear
IFMATCH '[cC]' IN $k THEN DISPLAYFILE idisp
IFMATCH '[dD]' IN $k THEN DISPLAYFILE idisps
IFMATCH '[eE]' IN $k THEN
   DISPLAYFILE iif1
   READ $dum
   DISPLAYFILE iif2
ENDIF
IFMATCH '[Ff]' IN $k THEN DISPLAYFILE iifm
IFMATCH '[gG]' IN $k THEN DISPLAYFILE imac
IFMATCH '[hH]' IN $k THEN DISPLAYFILE inew
IFMATCH '[iI]' IN $k THEN DISPLAYFILE ipaus
IFMATCH '[jJ]' IN $k THEN DISPLAYFILE iprint
IFMATCH '[kK]' IN $k THEN DISPLAYFILE iread
IFMATCH '[lL]' IN $k THEN DISPLAYFILE iclock
IFMATCH '[mM]' IN $k THEN DISPLAYFILE irstu
IFMATCH '[nN]' IN $k THEN DISPLAYFILE istop
IFMATCH '[oO]' IN $k THEN DISPLAYFILE ishell
IFMATCH '[pP]' IN $k THEN DISPLAYFILE iwstu
IFMATCH '[xX]' IN $k THEN
ASSIGN frame_1 TO $NEXTFRAME
ELSE
DISPLAY
DISPLAY Press return to continue
READ $k
CLEARSCREEN
ENDIF
```

"langtut" Frame in Menu-Driven Authoring System

```
COMMENT  langtut frame
COMMENT  invoked by frame_1

CLEARSCREEN
ASSIGN langinst TO $NEXTFRAME
DISPLAYFILE operate
DISPLAY Press 'RETURN' to continue.
READ $key
CLEARSCREEN
DISPLAYFILE langover
DISPLAY
DISPLAY Press 'RETURN' to continue.
READ $key
CLEARSCREEN
```

"lstdisp" Frame in Menu-Driven Authoring System

```
COMMENT  lstdisp frame
COMMENT  invoked by frammenu

CLEARSCREEN

DISPLAYFILE leadblnk

DISPLAY
DISPLAY  List of existing display files:
DISPLAY

SHELL ls -l ../$course/disp ¦ more

DISPLAY  Please hit return to continue:

READ $dum

ASSIGN frammenu TO $NEXTFRAME
```

"lstcours" Frame in Menu-Driven Authoring System

```
COMMENT lstcours frame
COMMENT invoked by lstcours

CLEARSCREEN

DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY  List of existing courses:
DISPLAY
DISPLAY

SHELL ls -l ../ | more

DISPLAY  Please hit return to continue:

READ $dum

ASSIGN courseme TO $NEXTFRAME
```

"lstframe" Frame in Menu-Driven Authoring System

```
COMMENT lstframe frame
COMMENT invoked by frammenu

CLEARSCREEN

DISPLAYFILE leadblnk

DISPLAY
DISPLAY  List of existing frame files:
DISPLAY

SHELL ls -l ../$course/frame | more

DISPLAY  Please hit return to continue:

READ $dum

ASSIGN frammenu TO $NEXTFRAME
```

"reenter" Frame in Menu-Driven Authoring System

```
COMMENT  reenter macro

DISPLAY
DISPLAY
DISPLAY  You entered > $choice < which was not among the valid
DISPLAY  menu selections. Please reenter a valid choice when
DISPLAY  presented with the menu again.
DISPLAY
DISPLAY  Please hit return to continue:

READ $dum
```

"rmcrs" Frame in Menu-Driven Authoring System

```
COMMENT rmcrs frame
COMMENT invoked by courseme

CLEARSCREEN
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY   Enter the name of the course to delete:
READ $delcrs
DISPLAY
DISPLAY   Are you sure?  Enter Y to continue.
READ $answr
ASSIGN courseme TO $NEXTFRAME
IFMATCH '^[yY]$'  IN $answr THEN ASSIGN zapcrs TO $NEXTFRAME
IFMATCH 'cbtcbt' IN $delcrs THEN
DISPLAY You cannot delete the authoring system course
ASSIGN courseme TO $NEXTFRAME
ENDIF
IFMATCH 'uset' IN $delcrs THEN
DISPLAY You cannot delete the authoring system
ASSIGN courseme TO $NEXTFRAME
ENDIF
SHELL if (test ! -d ../$delcrs) then
   DISPLAY  The course does not exist.
   ASSIGN courseme TO $NEXTFRAME
SHELL fi
```

"rmdsp" Frame in Menu-Driven Authoring System

```
COMMENT  rmdsp frame
COMMENT  invoked by frammenu

CLEARSCREEN
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY  Enter the name of the display file to delete:
READ $dnam
SHELL rm disp/$dnam
ASSIGN frammenu TO $NEXTFRAME
```

"rmfrm" Frame in Menu-Driven Authoring System

```
COMMENT  rmfrm frame
COMMENT  invoked by frammenu

CLEARSCREEN
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY Please enter then name of the frame to be deleted:
READ $fnam
SHELL rm frame/$fnam
ASSIGN frammenu TO $NEXTFRAME
```

"selcours" Frame in Menu-Driven Authoring System

```
COMMENT  selcours frame
COMMENT  invoked by courseme

CLEARSCREEN

DISPLAYFILE leadblnk

DISPLAY
DISPLAY  Please enter the course name that you want to author.
DISPLAY  Note that a course directory structure will be created
DISPLAY  if the course name entered does not exist.
DISPLAY
DISPLAY  Enter course name:

READ $course

IFMATCH '^$' IN $course THEN
   ASSIGN selcours TO $NEXTFRAME
   DISPLAY  You entered a null course name.
   DISPLAY  Please reenter when the display is refreshed.
   PAUSE 3

ELSE
   ASSIGN chkcours TO $NEXTFRAME
ENDIF
```

"zapcrs" Frame in Menu-Driven Authoring System

```
COMMENT zapcrs frame
COMMENT invoked by rmcrs

DISPLAY please wait ...
SHELL mvdir ../$delcrs ../oldjunk
ASSIGN courseme TO $NEXTFRAME
```

"ftran"

```
#         Frame  Translator
#
for i in $*
  do
    awk -f ../../uset/pre_fltr $i |
    awk -f ../../uset/mai_fltr - >../exef/$i
    echo
    echo Translation completed for frame -- $i.
    echo
  done
```

"matchtes

This frame has no initial content.

"matchtest"

This frame has no initial content.

"restore"

```
stty echo icanon icrnl opost ixon brkint isig
```

"setpaths"

```
PATH=$PATH:.:..:/usr/tmp/vri/uset
```

"fdrive"

```
#   Frame Driver
#
NEXTFRAME=frame_1
trap '. exef/escframe' 2
while (true)
do
   . exef/$NEXTFRAME
done
```

"frame_1"

```
COMMENT  Generic frame_1 frame
DISPLAY  This is the generic frame_1 frame. You will want to:
DISPLAY        1) remove these display lines, and
DISPLAY        2) assign your initial frame name to NEXTFRAME.
DISPLAY
```

"mai_fltr"

```
#
# Awk filters for frames
#
$1 == "ELSE" {if ($2 == "IF") {
                        printf "elif (test \"`expr "
                        for (i = 3; $i != "THEN"; i++)
                          printf "%s ",$i
                        printf "`\" -ne \"Ø\") then\n"
                               }
              if ($2 == "IFMATCH") {
                        printf "echo %s >../uset/matchtes\n",$5
                        printf "elif (test \"`grep -c %s",$3
                        printf " ../uset/matchtes `\" -ne \"Ø\")
then\n"
                                   }
              if ($2 == "") {
                        printf "else\n"
                            }
              next
             }
$1 == "ASSIGN" {printf "%s=`expr ",substr($NF,2)
                for (i = 2; $i != "TO"; i++)
                     printf "%s ",$i
                printf "`\n"
                next
               }
$1 == "CLEAR" {printf "> buf/%s",$2
               next
              }
$1 == "CLEARSCREEN" {$1 = "clear"
                     print
                     next
                    }
$1 == "DISPLAY" {printf "echo \n"
                 printf "echo '""
                 printf "%s", substr($Ø,length($1)+1+index($Ø,$1))
                 printf "\\c\"\n"
                 next
                }
$1 == "DISPLAYFILE" {printf "cat disp/%s\n",$2
                     next
                    }
$1 == "ENDIF" {$1 = "fi"
               print
               next
              }
```

```
$1 == "FREEPLAY" {printf "LOGNAME=root ; export LOGNAME\n"
                  printf "MACHINE='uname -m | sed -e
\"s;.*/\(.*\);\1;\"' ; export MACHINE\n"
                  printf "/usr/users/c3i/bin/c3iexec\n"
                  next
                 }
$1 == "GETDISPLAY" {printf "cat buf/%s",$2
                    next
                   }
$1 == "INTERACT" {printf "echo '"
                  printf $0
                  printf "'"
                  next
                 }
$1 == "KEYPRESS" {printf "echo '"
                  printf $0
                  printf "'"
                  next
                 }
$1 == "MACRO" {printf ". exef/%s\n",$2
               next
              }
$1 == "NEWSTU" {printf "if (test ! -d stupro/%s) then\n",$2
                printf "   mkdir stupro/%s\n",$2
                printf "fi\n"
                next
               }
$1 == "PAUSE" {$1 = "sleep"
               print
               next
              }
$1 == "PRINTFILE" {printf "lp disp/%s\n",$2
               next
              }
$1 == "READ" {$1 = "read"
              for (i = 2; i <= NF; i++)      #strip leading $'s from
vars
                  $i = substr($i,2,length($i)-1)
              print
              next
             }
$1 == "READCLOCK" {printf "sec=`date '+%S'`\n"
                   printf "min=`date '+%M'`\n"
                   printf "hr=`date '+%H'`\n"
                   printf "%s",substr($2,2)
                   printf "=`expr 3600 \\* $hr + 60 \\* $min + $sec`\n"

                   next
                  }
```

```
$1=="READSTU" {printf "%s=`cat stupro/%s/%s`\n",substr($6,2),$2,$4
                next
               }
$1 == "SAVEO" {printf "cat buf/obuf >> buf/%s",$2
               next
              }
$1 == "SCREEN" {printf "echo '"
                printf $0
                printf "'"
                next
               }
$1 == "SEND" {printf "echo '"
              printf $0
              printf "'"
              next
             }
$1 == "STOPCOURSE" {$1 = "exit 1"
                    print
                    next
                   }
$1 == "SHELL" {$1 = ""
              print
              next
             }
$1 == "WRITESTU" {printf "echo %s >stupro/%s/%s\n",$6,$2,$4
                  next
                 }
{print}
```

"pre_fltr"

```
# cat prefilter
#
# Awk filters for frames
#
$1 == "IFMATCH" {printf "echo %s >../uset/matchtes\n",$4
                 printf "if (test \"`grep -c %s",$2
                 printf " ../uset/matchtes `\" -ne \"0\") then\n"
                 if ($NF != "THEN") {
                    for (i = 6; i <= NF; i++)
                         printf "%s ",$i
                    printf "\nfi\n"
                 }
                 next
                }
$1 == "IF" {printf "if (test \"`expr "
            for (i = 2; $i != "THEN"; i++)
                 printf "%s ",$i
            printf "`\" -ne \"0\") then\n"
            ++i
            if (i < NF) {
               while (i <= NF) {
                  printf "%s ",$i
                  ++i
               }
               printf "\nfi\n"
            }
            next
           }
 $1 != "COMMENT" {print}
```

APPENDIX B

CREATION AND MODIFICATION OF CAI COURSEWARE

The training for database operations discussed was created at the same time that the authoring software was under development. Therefore, this training was not created using all the tools provided by the ultimate authoring language product. However, authors of future modules may find it easier to compose training using the menu-driven tool found in the authoring package. To do this, the authors will have to learn the authoring language.

Alternatively, authors may find it easier to copy some existing logic software and then to modify it slightly to fit their needs. This approach does not require as much knowledge of the authoring language, because some routines can be used as is, and changes can be made while the author carefully follows the existing command syntax. This method is discussed below.

A third approach is to author using the menu-driven tool, but to use the printed templates as a guide.

The following discussion uses the database training delivered with the authoring system as a baseline. Certain techniques, such as 'iow to allow an interruption, how to allow movement within the course, how to deal with questions, and so forth, can be handled in a number of ways. The methods used in the Database Training Course work, but they are not the only way to achieve acceptable results.

Overview of Course Software Structure

The course consists of a set of lessons (Lessons 1 through 6), each of which is a subdirectory. Each lesson subdirectory has several subdirectories, "disp," "frame," "exef," and "stupro." These names are identical in all lessons. The first three are of importance in courseware development and modification.

The "disp" subdirectory consists of a set of files, one file for each display to be presented on the screen. The "disp" frames contain the actual displays to be shown on the screen.

The "frame" subdirectory consists of a set of files, one file for each separate logic frame. In general, every logic frame calls for a separate display frame. These frames contain logic written in the authoring language.

The "exef" subdirectory contains a set of files, one file for each logic frame. These files are simply translations of the "frame" files. They are produced by the authoring software. It is the "exef" files that contain the UNIX code that actually is used during the running of the lesson.

Modification of Courseware

The current courseware can be modified by the Army to follow changes in the operation of the MCS 2 database, or to improve the quality of instruction following initial use and evaluation. There are two kinds of changes that may be required, simple changes to existing displays, and more complex changes to the training software.

Tools Required for Courseware Modification. All modifications can be made using a few basic UNIX commands, copy (cp), move (mv), remove (rm), make directory (mkdir) and remove directory (rmdir), and a UNIX editor such as vi. The vi editor is recommended for changes to displays, because it allows full screen viewing of displays, which will appear very much as they do in the course presentation. A basic knowledge of UNIX is required, so that the person performing the courseware modification can get to the subdirectories containing the frames requiring modification. Only a few vi commands are needed to start (i.e., i, a, x, cw, cW, dw, dW, dd, r, R, ZZ, and :q!), although greater knowledge may speed the task. The hints in this paragraph will require that the user obtain documentation or help on the UNIX operating system.

During the development of this courseware, adequate familiarization was possible with a day of instruction and a week of initial under supervision.

Changes to Displays. Some of the changes may entail only changing the content of one or a few displays. In this case, only editing of existing display files is necessary. These changes are easy to make, and are relatively risk-free, because the logic is unchanged. An error in the modification process will only cause an error in the display of a frame, not in the logic sequence and running of the rest of the lesson.

To make a change in a display the operator goes to the subdirectory containing the lesson to be modified (e.g., lesson1), and then to the "disp" subdirectory within the lesson subdirectory. The frame containing the display in question is edited using the vi editor. The person making changes should be careful to keep all displays to 80 columns or fewer, and to make all displays 22 lines long. This allows the trainee to press the return key, causing the cursor to go to the next line, without scrolling the display. Once the changes to a display frame have been made, the frame is saved, and the process is complete.

New Logic Frames. It is possible to write the logic for each new frame that is required. However, an alternative is to use existing frames as templates for new logic frames. If the new logic frames are supposed to do the same thing as existing frames, with different displays and different destinations after a keypress, then the easy way to make an addition is to copy an existing frame of the desired type to a new file, and then change only the addresses or pointers in this new file.

Changes to Logic. Each logic frame stands alone, and must contain within it the code for all key presses that are to be recognized, plus the name of the display frame to be displayed on the screen. The person making changes to logic must consider the following possible changes:

1. The name of the display frame to be presented must appear in the logic frame.

2. The name(s) of the next logic frame(s) to which the program may go at a keypress must be specified. This includes the backstep frame.

3. Other logic frames may be affected by the current change. When a new logic frame is created the logic frame which will immediately precede the new one must be changed so that the proper keypress calls for the new frame (this is usually a carriage return). The immediately subsequent frame must also be changed so the backstep keypress sends the program to the new logic frame (this is usually the "b" key.

To make a change in logic the person making the change should go to the desired lesson subdirectory (e.g., lessonl) and then to the "frame" subdirectory within the lesson subdirectory.

The recommended manner of adding a logic frame is: (1) copy a frame with similar structure to the new name, and (2) make changes to the addresses in the logic frame. In this way, the programming techniques developed by the original programmers will be transferred to the new frame.

Once the new frame has been developed, it must be translated from the authoring language into UNIX shell script. The translation is performed from the "frame" subdirectory of the lesson, where the operator types the command "sh ../../uset/ftran [name of the logic frame]", (e.g., sh ../../uset/ftran frame_1). The authoring software makes the translation and puts the resulting executable frame into the "exef" subdirectory in the same lesson. When the courseware is run, the actual commands come from the files in the "exef" subdirectory.

Appendix A contains several sample files and indications of how files can be altered.

Working With the "OOT" Script. If the training package is to use the "oot" script for initial selection, then a subdirectory must be created within the lesson, called "oldstudent." This subdirectory must contain four files, each of which must contain the following data:

1. File "correct," containing the value Ø.

2. File "dummy," containing the word "dummy."

3. File "place," containing the character ~.

4. File "qasked," containing the value Ø.

An identical "oldstudent" subdirectory should be created within stupro.

In addition, the "OOT" file should be modified to allow the selection of the new lesson.

Current Topic. In the delivered training, the trainee has the opportunity to jump back to the beginning of the topic that he is working on. In order to do this, a new topic must be assigned as the trainee reaches the first frame of the topic. This requires a statement of the form: ASSIGN [filename] TO $TOPIC. Henceforth, $TOPIC contains the name of the current topic. This must be done at each frame where the topic changes. If the student selects a new topic in TOPICFRAME, the code for that new frame must contain the reassignment of $TOPIC. This will be the case if the new frame is the first frame of the new topic.

APPENDIX C

TEMPLATES OF AUTHORING CODE IMPLEMENTATION FOR SELECTED SCREEN PRESENTATIONS

Template for a Frame That Displays for a Fixed Number of Seconds

```
CLEARSCREEN
ASSIGN {display frame name} TO $DISPLAY
DISPLAYFILE $DISPLAY
ASSIGN {current frame name} TO $NEXTFRAME

COMMENT Pause {no. of seconds}
PAUSE 2
ASSIGN {next frame name} TO $NEXTFRAME
```

Template for a Standard Frame

```
COMMENT This frame presents information until the trainee presses
COMMENT <Return>
COMMENT It also assigns a new frame address to $TOPIC (the current
COMMENT topic)
CLEARSCREEN

COMMENT Keep track of the current display, so it can be printed if
COMMENT necessary.
ASSIGN {display frame name} TO $DISPLAY
COMMENT Display the proper file.
DISPLAYFILE $DISPLAY

COMMENT Wait for a keypress.
READ $Keypress

COMMENT Assign the current frame to $NEXTFRAME
ASSIGN {current frame name} TO $NEXTFRAME

COMMENT Assign a new frame address to $TOPIC.
ASSIGN {current frame name} TO $TOPIC

COMMENT Look for a carriage return with no other input ('^$').
IFMATCH '^$' IN $Keypress THEN ASSIGN {next frame name} TO $NEXTFRAME

COMMENT Go back to {previous frame name}
IFMATCH '[Bb]$' IN $Keypress THEN ASSIGN {previous frame name} TO $NEXTFRAME

COMMENT "serveout" is a MACRO.  It has some fixed code that is identical in
COMMENT many frames.  This code allow the trainee to branch to an interrupt
COMMENT routine to select a new topic, print, quit, etc.  Rather than add
COMMENT this code to every frame, MACRO serveout is invoked.
MACRO serveout
```

Templates for a Quiz Question With Responses

Question Root Frame

```
COMMENT Quiz Frame
CLEARSCREEN
ASSIGN {question display frame} TO $DISPLAY

COMMENT Store the display address in case of print request.
DISPLAYFILE $DISPLAY

COMMENT The next assignment is not necessary, but it is a place to
COMMENT write the name of the current frame so the author can refer to
COMMENT it.
ASSIGN {current frame name} TO $NEXTFRAME

COMMENT Read and test student response.  Choices are 1 through 4.
READ $Keypress
 IFMATCH [1] IN $Keypress THEN ASSIGN {choice 1 frame} TO $NEXTFRAME
 IFMATCH [2] IN $Keypress THEN ASSIGN {choice 2 frame} TO $NEXTFRAME
 IFMATCH [3] IN $Keypress THEN ASSIGN {choice 3 frame} TO $NEXTFRAME
 IFMATCH [4] IN $Keypress THEN ASSIGN {choice 4 frame} TO $NEXTFRAME

COMMENT Allow an interrupt.
MACRO serveout
```

Templates for a Quiz Question With Responses

Frame for a Wrong Answer
(There are usually three of these per question)

```
COMMENT Wrong Answer Frame
CLEARSCREEN

COMMENT Store the display address in case of print request.
ASSIGN {answer display frame} TO $DISPLAY
DISPLAYFILE $DISPLAY

COMMENT The next assignment is not necessary, but it is a place to
COMMENT write the name of the current frame so the author can refer to
COMMENT it.
ASSIGN {current frame name} TO $NEXTFRAME

COMMENT The next line shows that the trainee missed the particular item on
COMMENT this quiz one more time.
IF $wrong = Ø THEN
   ASSIGN $qmissed + 1 TO $qmissed
   ASSIGN 1 TO $wrong
ENDIF

COMMENT The student must press <Return> to continue.
READ $Keypress
IFMATCH '^$' IN $Keypress THEN ASSIGN {question root frame} TO $NEXTFRAME

COMMENT Allow an interrupt.
MACRO serveout
```

Templates for a Quiz Question With Responses

Frame for a Correct Answer

```
COMMENT Correct Answer Frame

COMMENT Increment the number of questions asked.
ASSIGN $qasked + 1 TO $qasked

COMMENT Clear the $wrong flag.
ASSIGN Ø TO $wrong

CLEARSCREEN
COMMENT Store the display address in case of print request.
ASSIGN {answer display frame} TO $DISPLAY
DISPLAYFILE $DISPLAY

COMMENT The next assignment is not necessary, but it is a place to
COMMENT write the name of the current frame so the author can refer to
COMMENT it.
ASSIGN {current frame name} TO $NEXTFRAME

COMMENT The student must press <Return> to continue.
READ $Keypress
IFMATCH '^$' IN $Keypress THEN ASSIGN {next question frame} TO $NEXTFRAME

COMMENT Allow an interrupt.
MACRO serveout
```

Error Frame Templates

First Frame to Come Up When an Error Is Detected

```
COMMENT Save the name of the last frame.
ASSIGN $NEXTFRAME TO $HOLD

COMMENT Now go to the error frame.
ASSIGN errorframe TO $NEXTFRAME
```

"errorframe" Template

```
COMMENT Display the message to be shown on an error.
DISPLAY You pressed the wrong keys!!  Look at the directions and try again.

COMMENT Return to original frame following any keyboard input.
ASSIGN $HOLD TO $NEXTFRAME
READ $Keypress
```

"CONTINUE" Frame Template

```
COMMENT This is the continue frame after a quiz.
CLEARSCREEN
COMMENT Compute number of questions correct
ASSIGN $qasked - $qmissed TO $CORRECT
ASSIGN $CORRECT \* 100 / $qasked TO $PCNT

COMMENT Give student feedback on performance to this point.
DISPLAY
DISPLAY
DISPLAY Lesson feedback for $student.
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY Thus far in the lesson you have correctly answered $CORRECT out of
DISPLAY $qasked questions ($PCNT%).
DISPLAY
DISPLAY When you are ready, press return to continue with the lesson.
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY Press <Return>
DISPLAY

COMMENT Reset the $CORRECT flag
ASSIGN 0 TO $CORRECT

COMMENT The student must press <Return> to continue
READ $Keypress
IFMATCH '^$' IN $Keypress THEN ASSIGN c1-01-05-00 TO $NEXTFRAME
```

"ENDFRAME" Template

```
COMMENT ENDFRAME is invoked when the student quits during the lesson.
CLEARSCREEN
ASSIGN Ø TO $leave
COMMENT Compute number correct.
ASSIGN $qasked - $qmissed TO $CORRECT
COMMENT Save questions asked, questions correct, and last topic in
COMMENT subdirectory $stupro (the student's name) in directory stupro.
WRITESTU $student ATTRIBUTE qasked VALUE $qasked
WRITESTU $student ATTRIBUTE correct VALUE $CORRECT
WRITESTU $student ATTRIBUTE place VALUE $TOPIC
COMMENT Give end of session feedback.
DISPLAY
DISPLAY
DISPLAY Feedback for $student
DISPLAY
DISPLAY
DISPLAY
DISPLAY You have chosen to discontinue the lesson.  Up to this point you
DISPLAY have correctly answered $CORRECT out of $qasked questions.
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY Press <Return> to end the lesson.
DISPLAY
READ $Keypress
IFMATCH '^$' IN $Keypress THEN ASSIGN 1 TO $leave
COMMENT STOPCOURSE is an authoring language command to stop the course.
IF $leave = 1 THEN
 CLEARSCREEN
 STOPCOURSE
ENDIF
```

"LASTFRAME" Template

```
COMMENT This frame the results of the questions in $stupro in the
COMMENT stupro directory.

COMMENT Read the clock to set the finishing time.
READCLOCK $FINISHTIME
COMMENT Compute time elapsed.
ASSIGN $FINISHTIME - $STARTTIME TO $TOTALTIME
ASSIGN $TOTALTIME / 60 TO $TOTALMINUTES

COMMENT Compute number of questions correct.
ASSIGN $qasked - $qmissed TO $CORRECT
COMMENT Store values in $stupro in the stupro directory.
WRITESTU $student ATTRIBUTE qasked VALUE $qasked
WRITESTU $student ATTRIBUTE correct VALUE $CORRECT

COMMENT Note that the student finished this lesson.
WRITESTU $student ATTRIBUTE place VALUE done

COMMENT Give end of lesson feedback.
CLEARSCREEN
DISPLAY
DISPLAY Lesson feedback for $student.
DISPLAY
DISPLAY
DISPLAY Congratulations, you have successfully completed Lesson XXX!
DISPLAY
DISPLAY
DISPLAY
DISPLAY You correctly answered $CORRECT out of $qasked questions
DISPLAY and took $TOTALMINUTES minutes to complete the lesson.
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY
DISPLAY Press <Return> or <I>
DISPLAY
READ $Keypress

COMMENT STOPCOURSE is an authoring language command to stop the course.
IFMATCH '^$' IN $Keypress THEN
 CLEARSCREEN
 STOPCOURSE
ENDIF
```

"OUTFRAME" Frame Template

```
COMMENT The trainee may select what he wants to do at this interruption.
CLEARSCREEN
DISPLAYFILE dOUTFRAME
READ $Keypress
ASSIGN OUTFRAME TO $NEXTFRAME

COMMENT Choice 1 is to return to where the trainee was.
IFMATCH '[1]$' IN $Keypress THEN ASSIGN $HOLDFRAME TO $NEXTFRAME

COMMENT Choice 2 goes back to the beginning of this topic.
IFMATCH '[2]$' IN $Keypress THEN ASSIGN $TOPIC TO $NEXTFRAME

COMMENT Choice 3 goes to the TOPICFRAME frame to allow selection of a
COMMENT new topic.
IFMATCH '[3]$' IN $Keypress THEN ASSIGN TOPICFRAME TO $NEXTFRAME

COMMENT Choice 4 goes to ENDFRAME to end the lesson.
IFMATCH '[4]$' IN $Keypress THEN ASSIGN ENDFRAME TO $NEXTFRAME

COMMENT Choice 5 requests that Unix print the display.
IFMATCH '[5]$' IN $Keypress THEN PRINTFILE $DISPLAY

COMMENT Clear the screen following a keypress.
CLEARSCREEN
```

"TOPICFRAME" Template

```
COMMENT The trainee may select from a menu of topics to study.
CLEARSCREEN
DISPLAYFILE dTOPICFRAME
READ $Keypress
COMMENT The Topic# references the frame which begins the topic
COMMENT selected by the trainee.
IFMATCH '[1]$' IN $Keypress THEN ASSIGN c1-ØØ-Ø1-ØØ TO $NEXTFRAME
IFMATCH '[2]$' IN $Keypress THEN ASSIGN c1-Ø1-Ø1-ØØ TO $NEXTFRAME
IFMATCH '[3]$' IN $Keypress THEN ASSIGN c1-Ø1-Ø5-ØØ TO $NEXTFRAME
IFMATCH '[4]$' IN $Keypress THEN ASSIGN c1-Ø1-Ø6-ØØ TO $NEXTFRAME
COMMENT This choice ends the session.
IFMATCH '[eE]$' IN $Keypress THEN ASSIGN ENDFRAME TO $NEXTFRAME
```

"frame_1" Template

```
CLEARSCREEN
COMMENT Assign zero to working variables
ASSIGN Ø TO $qasked
ASSIGN Ø TO $qmissed
ASSIGN Ø TO $STARTTIME
ASSIGN Ø TO $FINISHTIME
ASSIGN Ø TO $TOTALTIME
ASSIGN Ø TO $TOTALMINUTES
ASSIGN Ø TO $CORRECT
ASSIGN Ø TO $wrong
ASSIGN Ø TO $PCNT

COMMENT Read the clock to time the student for this session.
READCLOCK $STARTTIME
COMMENT The initial display is contained in this frame.
DISPLAY
DISPLAY
DISPLAY
DISPLAY Please enter your last name:

COMMENT $student contains the student name under which data will be
COMMENT stored in stupro.
READ $student

COMMENT Set up a stupro subdirectory with this name.
NEWSTU $student
WRITESTU $student ATTRIBUTE dummy VALUE dummy

COMMENT Copy this student's prior record to oldstudent.
SHELL cp stupro/$student/* stupro/oldstudent

COMMENT Set up a clean set of files under the student's name.
WRITESTU $student ATTRIBUTE qasked VALUE Ø
WRITESTU $student ATTRIBUTE correct VALUE Ø
WRITESTU $student ATTRIBUTE place VALUE ~

COMMENT Copy oldstudent back to the account.
SHELL cp stupro/oldstudent/* stupro/$student

COMMENT If there is a value in the place field, then the student left
COMMENT the lesson in the middle.  Start where he/she left off.  This
COMMENT is the beginning of the last TOPIC that the student started.
READSTU $student ATTRIBUTE place VALUE $place
COMMENT Set $NEXTFRAME and $TOPIC to the value they had when the
COMMENT student ended the last session.
ASSIGN $place TO $TOPIC
ASSIGN $place TO $NEXTFRAME
```

```
COMMENT If there is a ˜ in $place then the student did not leave
COMMENT in a previous session.  Start at the beginning.
IFMATCH '˜$' IN $place THEN
     DISPLAY Welcome to the course.
     COMMENT Set $NEXTFRAME and $TOPIC to the first frame.
     ASSIGN c1-ØØ-Ø1-ØØ TO $NEXTFRAME
     ASSIGN c1-ØØ-Ø1-ØØ TO $TOPIC
ENDIF

COMMENT If there is a "done" in $place then this student completed the
COMMENT course once.  Restart at the beginning, and keep the old data
COMMENT in a subdirectory with the student's name and the extension
COMMENT ".old".
IFMATCH 'done$' IN $place THEN
DISPLAY You have completed Lesson 1.  You will restart at the beginning.
COMMENT Set $NEXTFRAME and $TOPIC to the first frame.
     ASSIGN c1-ØØ-Ø1-ØØ TO $NEXTFRAME
     ASSIGN c1-ØØ-Ø1-ØØ TO $TOPIC
     COMMENT Keep the old data.
     NEWSTU $student.old
     READSTU $student ATTRIBUTE qasked VALUE temp
     WRITESTU $student.old ATTRIBUTE qasked VALUE temp
     READSTU $student ATTRIBUTE correct VALUE temp
     WRITESTU $student.old ATTRIBUTE correct VALUE temp
     WRITESTU $student ATTRIBUTE qasked VALUE Ø
     WRITESTU $student ATTRIBUTE correct VALUE Ø
     WRITESTU $student ATTRIBUTE place VALUE ˜
ENDIF

COMMENT Clear the temporary subdirectory oldstudent.
WRITESTU oldstudent ATTRIBUTE qasked VALUE Ø
WRITESTU oldstudent ATTRIBUTE place VALUE ˜
WRITESTU oldstudent ATTRIBUTE correct VALUE Ø

COMMENT Initialize working variables from the $student subdirectory
READSTU $student ATTRIBUTE qasked VALUE $qasked
READSTU $student ATTRIBUTE correct VALUE $CORRECT
ASSIGN $qasked - $CORRECT TO $qmissed
```

"serveout " Frame Template

```
COMMENT I or i will generate an interrupt wherever this MACRO is used.
IFMATCH '[iI]$' IN $Keypress THEN
 ASSIGN $NEXTFRAME TO $HOLDFRAME
 ASSIGN OUTFRAME TO $NEXTFRAME
ENDIF
```

"oot" Program

This script allows a menu-driven selection of activities following initiation of the script. It selects one of six lessons, or allows printout of student results or erasure of the student profile subdirectory. The script is written in Unix command language.

```
valid_input=0
while (test $valid_input -eq 0)
do
   clear
   cat Firstchoice
   read user_input

   case "$user_input" in

      1)
      cd lesson1
      sh .. uset fdrive
      cd ..
      ;;

      2)
      cd lesson2
      sh .. uset fdrive
      cd ..
      ;;

      3)
      cd lesson3
      sh .. uset fdrive
      cd ..
      ;;

      4)
      cd lesson4
      sh .. uset fdrive
      cd ..
      ;;

      5)
      cd lesson5
      sh ... uset fdrive
      cd ..
      ;;

      6)
      cd lesson6
      sh ... uset fdrive
      cd ..
      ;;
```

On this page the script continues. These inputs (+1 to +6) generate a printout of student profiles.

```
+1)
cd lesson1 stupro
{
    for file_name in *
    do
        echo $file_name
        cat $file_name *
    done
} | tee ../printout | more
read dummy
lp ../printout
read dummy
cd ../..
;;

+2)
cd lesson2 stupro
{
    for file_name in *
    do
        echo $file_name
        cat $file_name *
    done
} | tee ../printout | more
read dummy
lp ../printout
read dummy
cd ../..
;;

+3)
cd lesson3 stupro
{
    for file_name in *
    do
        echo $file_name
        cat $file_name *
    done
} | tee ../printout | more
read dummy
lp ../printout
read dummy
cd ../..
;;
```

```
+4)
 cd lesson4 stupro
 {
     for file_name in *
     do
          echo $file_name
          cat $file_name *
     done
 } | tee ../printout | more
 read dummy
 lp ../printout
 read dummy
 cd ../..
 ;;

 +5)
 cd lesson5 stupro
 {
     for file_name in *
     do
          echo $file_name
          cat $file_name *
     done
 } | tee ../printout | more
 read dummy
 lp ../printout
 read dummy

 cd ../..
 ;;

 +6)
  cd lesson6 stupro
 {
     for file_name in *
     do
          echo $file_name
          cat $file_name *
     done
 } | tee ../printout | more
 read dummy
 lp ../printout
 read dummy
 cd ../../
 ;;
```

The following lines allow the training manager to erase student profiles.

```
-1)
echo "Do you want to remove all student records for Lesson 1?"
read confirm
if test $confirm = y
then {
     cd lesson1
     rm -r stupro/*
     mkdir stupro/oldstudent
     cp oldstudent/* stupro/oldstudent
cd ..
}
fi
;;

-2)
echo "Do you want to remove all student records for Lesson 2?"
read confirm
if test $confirm = y
then
{
     cd lesson2
     rm -r stupro/*
     mkdir stupro/oldstudent
     cp oldstudent/* stupro/oldstudent
cd ..
}
fi
;;

-3)
echo "Do you want to remove all student records for Lesson 3?"
read confirm
if test $confirm = y
then
{
     cd lesson3
     rm -r stupro/*
     mkdir stupro/oldstudent
     cp oldstudent/* stupro/oldstudent
cd ../
}
fi
;;
```

```
-4)
echo "Do you want to remove all student records for Lesson 4?"
read confirm
if test $confirm = y
then
{
     cd lesson4
     rm -r stupro *
     mkdir stupro oldstudent
     cp oldstudent * stupro oldstudent
cd ..
}
fi
;;

-5)
echo "Do you want to remove all student records for Lesson 5?"
read confirm
if test $confirm = y
then
{
     cd lesson5
     rm -r stupro *
     mkdir stupro oldstudent
     cp oldstudent * stupro oldstudent
cd ..
}
fi
;;

-6)
echo "Do you want to remove all student records for Lesson 6?"
read confirm
if test $confirm = y
then
{
     cd lesson6
     rm -r stupro/*
     mkdir stupro oldstudent
     cp oldstudent * stupro/oldstudent
cd ../
}
fi
;;
```

"oot" Program Continued

These lines allow the student to quit the program, and stop training.

```
        q)
        valid_input=1
        echo ""
        ;;

        *)
        echo " "
        sleep 5
        ;;
    esac
done
```

END